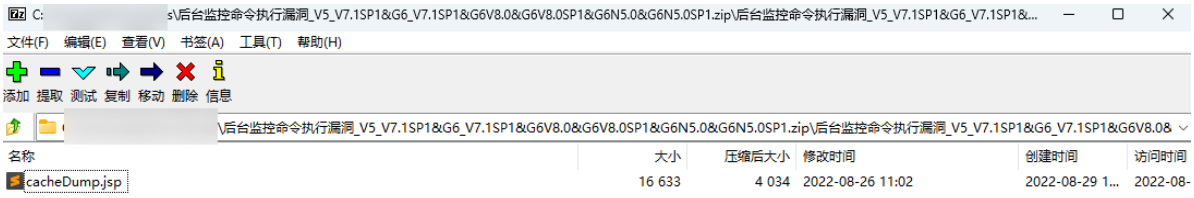
X远OA 管理员后台任意代码执行漏洞

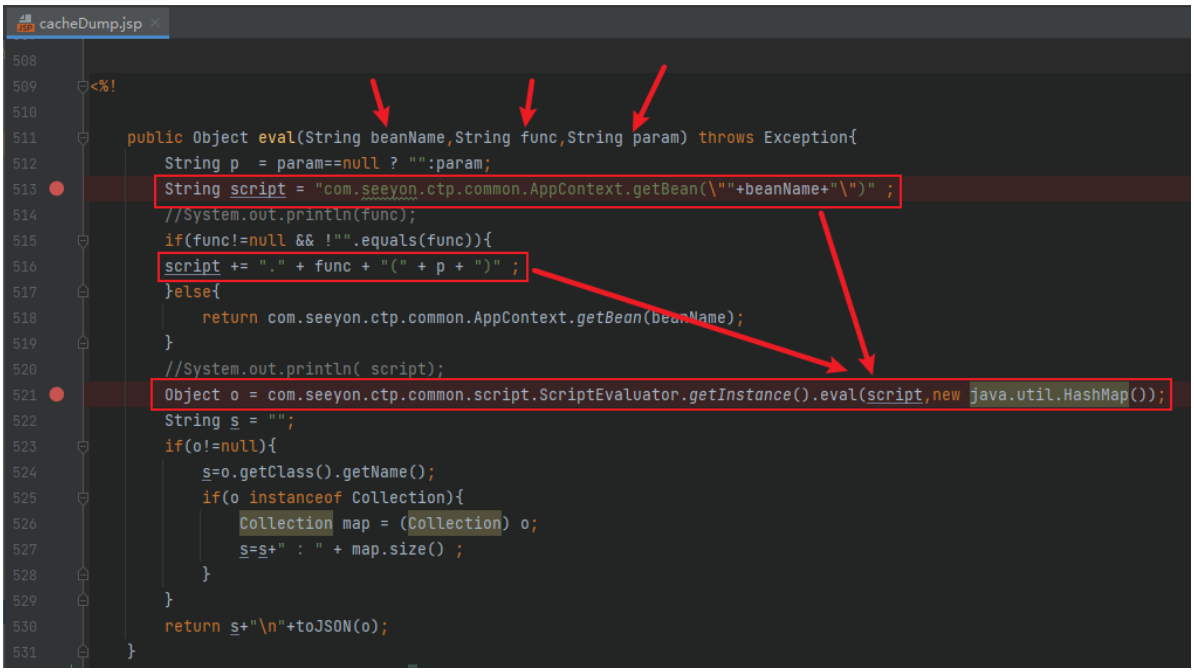
在X远OA的某个补丁中更新了一个文件



关键的代码处使用了 `ScriptEvaluator.getInstance().eval()`

定位到了 `cacheDump.jsp` 文件,该页面位于 `seeyon\web\WEB-INF\jsp\ctp\sysmgr\monitor\cacheDump.jsp`

发现其直接进行了代码的拼接，并进行了执行



噢？ 很经典的任意代码执行

那么我们再看看哪里调用了该函数

发现在同一文件中，对beanName进行了判断，若不等于onlineRecord，则判断一下methodName是不是已get开头的，若是，则传入eval

```
599  <%
600
601      if("onlineRecord".equals(beanName)){
602          Map all = getAllOnlineUser();
603          out.println(all.size());
604          out.println(toJSON(all));
605      }else{
606
607          if(methodName.startsWith("get")||methodName.equals("")){
608              out.println(eval(beanName,methodName,param));
609          }else{
610              out.println("只能执行名称为getXXX的方法。");
611          }
612      }
613
614  %>
```

这里出现了几个变量 beanName，methodName，param

去看一下这几个变量是否可控，发现在同一页面内，直接使用getParameter来接收这几个参数，至此绝对有理由怀疑该页面存在代码执行漏洞。

```
<%
    String beanName = request.getParameter("b");
    String methodName = request.getParameter("m");
    String param = request.getParameter("p");
%>
```

想要进入代码执行的部分，还有一个判断，当时差点看漏了

```
<%if("groovy".equals(q)){%>
<%...%>
<%!

    public Object eval(String beanName,String func,String param) throws Exception{
        String p  = param==null ? "":param;
        String script = "com.seeyon.ctp.common.AppContext.getBean(\""+beanName+"\")" ;
        //System.out.println(func);
        if(func!=null && !"".equals(func)){
        script += "." + func + "(" + p + ")" ;
        }else{
            return com.seeyon.ctp.common.AppContext.getBean(beanName);
        }
        //System.out.println( script);
        Object o = com.seeyon.ctp.common.script.ScriptEvaluator.getInstance().eval(script,new java.util.HashMap());
        String s = "";
        if(o!=null){
            s=o.getClass().getName();
            if(o instanceof Collection){
                Collection map = (Collection) o;
                s=s+" : " + map.size() ;
            }
        }
        return s+"\n"+toJSON(o);
    }
    public String toJSON(Object o){
        XStream xstream = new XStream((JsonHierarchicalStreamDriver) createWriter(writer) -> {
                return new JsonWriter(writer, JsonWriter.DROP_ROOT_MODE);
        });
        xstream.aliasSystemAttribute(null, "class");
        //xstream.aliasSystemAttribute(null, "defined-in");

        return xstream.toXML(o);
    }
```

而这个q也是我们可控的

```
<%
    String q = request.getParameter("q");
```

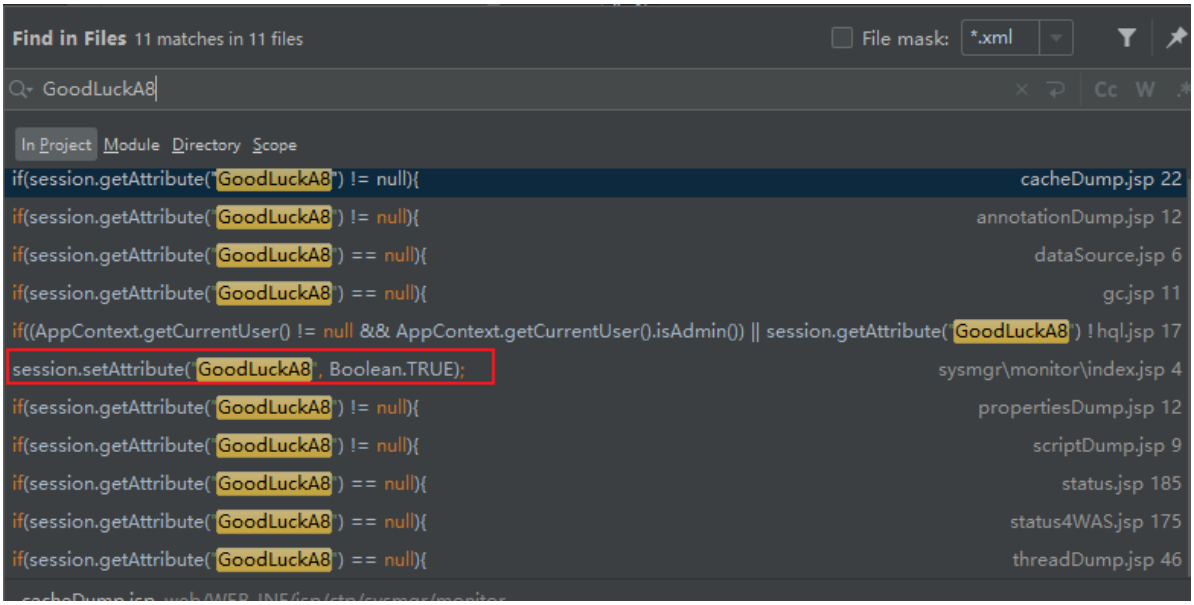
那么我们是否能够访问到该页面呢，在页面顶部有这样一段代码，

判断session中是否有`GoodLuckA8`

```
<%
if(session.getAttribute("GoodLuckA8") != null){

}
else{
    response.sendError(404);
    return;
}
%>
```

全局搜索`GoodLuckA8`,发现setAttribute操作



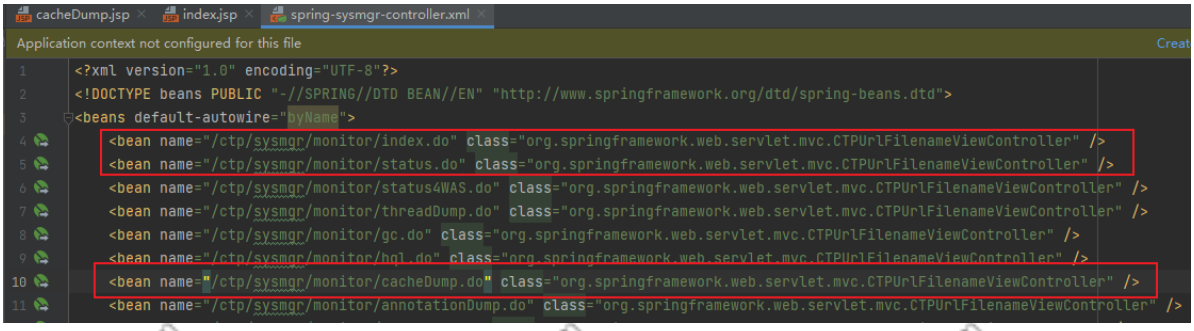
在这个页面中设置了`GoodLuckA8`为true，该页面位于`seeyon\web\WEB-INF\jsp\ctp\sysmgr\monitor\index.jsp`，也就是只要我们能够访问到这个页面，我们就能访问到能存在漏洞的页面。注意，这里由于服务器的容器不同，跳转的URL也会有所不同。

```
<%
    session.setAttribute("GoodLuckA8", Boolean.TRUE);

    String serverInfo = application.getServerInfo();

    if (serverInfo != null  && (serverInfo.contains("WebSphere") || serverInfo.contains("WebLogic") || serverInfo.contains("TongWeb"))) {
        response.sendRedirect("/seeyon/ctp/sysmgr/monitor/status4WAS.do");
    }
    else{
        response.sendRedirect("status.do");
    }
%>
```

随即去找此页面真正的路由，在`seeyon\web\WEB-INF\cfgHome\spring\spring-sysmgr-controller.xml`处发现了他们真正的访问地址。



进入它们指向的包查看一下，这里的访问的权限，必须是SystemAdmin 或者是 AccountAdministrator 权限的，

所以该页面是需要超级管理员权限才能访问的，不过之前也爆出过绕过系统管理员漏洞，可以接着这个漏洞使用。

```java
package org.springframework.web.servlet.mvc;

import ...

public class CTPUrlFilenameViewController extends UrlFilenameViewController {
    public CTPUrlFilenameViewController() {
    }

    protected String getViewNameForRequest(HttpServletRequest request) {
        String uri = this.extractOperableUrl(request);
        if (uri.indexOf("expscript.do") != -1 || uri.indexOf("sysmgr/monitor/") != -1) {
            User user = AppContext.getCurrentUser();
            Set<String> userRoles = (Set)user.getProperty("userRoles");
            if (userRoles == null || !userRoles.contains("SystemAdmin") && !userRoles.contains("AccountAdministrator")) {
                return this.getViewNameForUrlPath( uri: "404");
            }
        }

        return this.getViewNameForUrlPath(uri);
    }
}
```

接着就是构造payload了

通过上面的分析，我们需要以下几个步骤

```
1、登录获取超级管理员权限
2、访问 /seeyon/ctp/sysmgr/monitor/index.do 从而获取session
3、对 /seeyon/ctp/sysmgr/monitor/cacheDump.do 发送恶意payload
```

那么问题来了，cacheDump.do的参数应该怎么构造呢，还得继续看代码

要走到代码执行处，要经历这几步

```
beanName 对应 b
methodName 对应 m
param 对应 p
q 要等于 groovy
b 不能等于 onlineRecord
m 必须以 get开头
代码拼接
com.seeyon.ctp.common.AppContext.getBean(" + b + ").+ m + ( + p + )
```

所以这里最适合插入代码的就是p参数了

而这句代码的意思，就是获取一个Bean类，m必须get开头，要知道Bean类中都是有很多的getter和setter的方法的，而且取的这个方法，最好传的参数是一个int值和String值，以便我们进行拼接。

X远里面有很多这种开头的方法，自己找一下就可以发现了。