不安全的中间件 - Tomcat - SecPulse.COM | 安全脉搏

> 在 web 安全中，中间件安全也是非常重要的一部分，而中间件的安全问题主要来自两个方面：一个是中间件本身的开发缺陷导致的安全问题，另一个是开发或运维人员在使用时进行了错误的配置而导致的安全问题。

在 web 安全中，中间件安全也是非常重要的一部分，而中间件的安全问题主要来自两个方面：一个是中间件本身的开发缺陷导致的安全问题，另一个是开发或运维人员在使用时进行了错误的配置而导致的安全问题。Tomcat 也是众所周知的，应用最广泛的中间件了，那么接下来，我们就针对 Tomcat 中间件进行常见的安全问题及风险进行总结归纳。

版本管理

Tomcat 是属于 Apache 下的一个项目分支，类似于这种的软件项目，官方一般都会同时维护多个版本分支，一般新的产品特性会被更新在最新的大版本中，对于修复 bug 及漏洞这种就会在旧版本的分支中进行更新，这样就允许开发或运维人员在不破坏原有的生产环境的情况下完成对当前版本软件的更新。

例如当前你使用的 Tomcat 版本是 6.0.20，当需要进行升级更新时，在 6.0 版本分支中寻找新的版本（如 6.0.25），升级到最新的漏洞修复版本，如果在性能、功能等其他方有新的需求时，没有必要升级到 Tomcat7 版本。

对于 Tomcat 的使用者及维护者来说，就应该密切关注 Tomcat 官网中对安全漏洞和新版本的发布公告 (https://tomcat.apache.org/security.html) 及时了解漏洞信息及新版本的更新信息，这样就能及时判断自己所使用的版本是否存在安全隐患，是否需要更新。

运行环境

首先我们必须保证运行 Tomcat 的用户权限不能是高权限，比如 windows 下的 administrator 和 Linux 下的 root 用户或用户组，建议在使用 Tomcat 时创建一个 Tomcat 专属用户，在保证不影响业务正常使用的情况下将该用户权限降至最低，此外还要根据业务需求来对应用涉及的文件目录文件夹的读取、写入及执行的权限进行详细的分配。这样一来，就能很大程度上增加攻击者的攻击成本。

安全配置

Example 应用

Tomcat 在安装部署后，在 webapps 默认存在一个 examples 目录，该目录正如其文件名一样，提供一些示例应用让使用者来了解 Tomcat 的特性及功能。这些样例在业务上线后并没有什么用处，建议部署 tomcat 后，删除其中的样例文件（ROOT, balancer,jsp-examples, servlet-examples, tomcat-docs, webdav），避免信息泄露和其他潜在的安全风险。

这些样例中的 session 样例（/examples/servlets/servlet/SessionExample）允许用户对 session 进行操纵，因为 session 是全局通用的，所以用户可以通过操纵 session 获取管理员权限，存在一定的安全风险，不过这种基本上只有在一些比较老的不安全系统中才有可能出现，利用条件比较苛刻。

我们编写 3 个页面来模拟一般网站身份验证的过程。

login.jsp

```
<formaction=login2.jsp method="POST" >
用户名: <input type="text"name="username"><br>
密码: <input type="text" name="password"><br>
<inputtype="submit" value="登录"><br>
<form>
```

login1.jsp

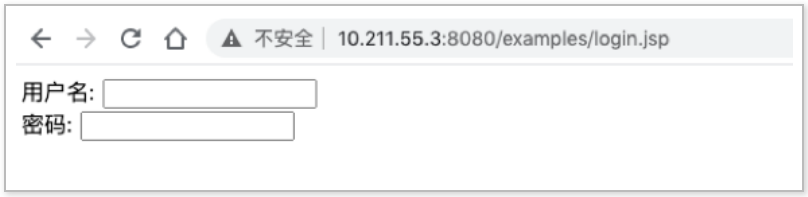
```
<%
if(request.getParameter("username") != null && request
   String username =request.getParameter("username");
   String password =request.getParameter("password");
      //验证身份
   if (username.equals("admin")&& password.equals("adm
      session.setAttribute("login","admin");
      response.sendRedirect("index.jsp");
   }else {
```

```
        response.sendRedirect("login.jsp");
    }
}
%>
```

index.jsp

```
<%
if(session.getAttribute("login")!= null &&((String)ses
   out.println("Login");
} else{
    response.sendRedirect("login.jsp");
}
%>
```

我们将写好的三个页面部署到 tomcat 上，我们首先访问一下 index.jsp 页面看看，访问之后跳转至 login.jsp。



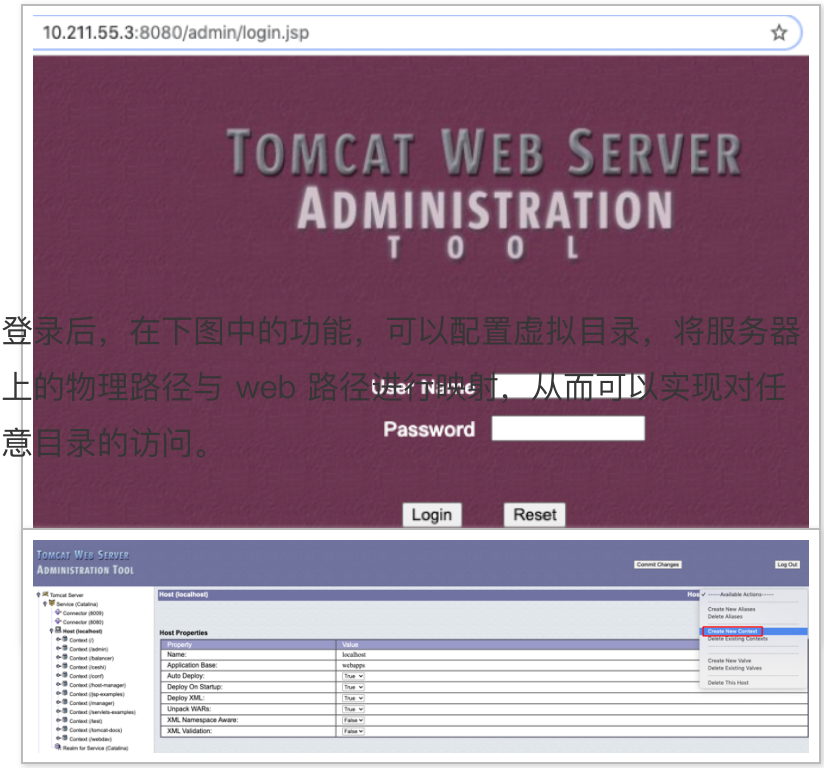
我们利用 examples 的 session servlet 功能操作一下 session，将 login 的值改成 admin，生成 session。

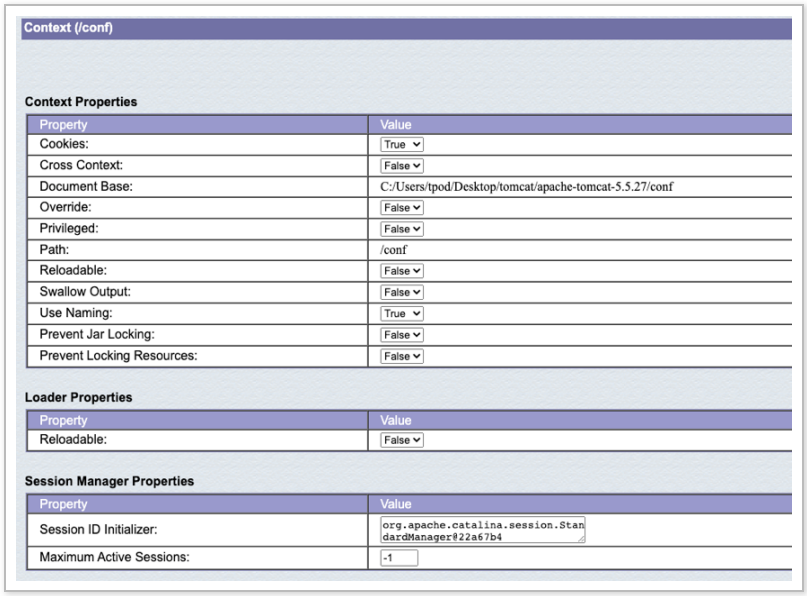
提交成功之后我们再来访问一下 index.jsp，发现不会跳转，直接限制 Login，说明我们生成的 session 有效，通过操作 session 绕过了登录。

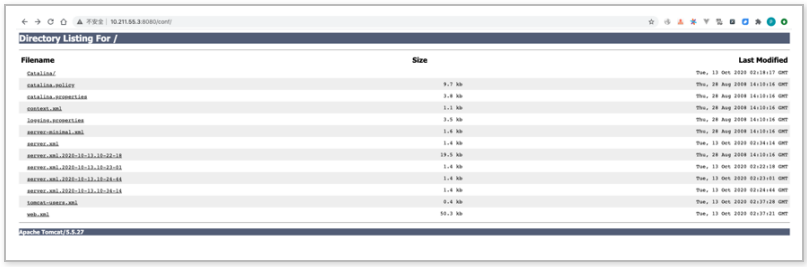
Admin 管理页面

Tomcat 的 admin console 全称是 Tomcat WEB Server Administration ToolT，该模块在 Tomcat 5.0.4 版本之前都是默认安装的，5.0.4 之后版本默认存在该目录，但是功能并不全，直接使用，从 5.5 版本开始作为可选模块进行安装，安装后的默认路径为 / admin，与 manager 配置相同，在 tomcat-user.xml 文件中进行账号密码的配置。该模块实现了通过 web 方式对 tomcat 服务、已部署的应用程序、连接池和其他资源的管理，方便运维及开发人员的管理和操作。

登录后，在下图中的功能，可以配置虚拟目录，将服务器上的物理路径与 web 路径进行映射，从而可以实现对任意目录的访问。







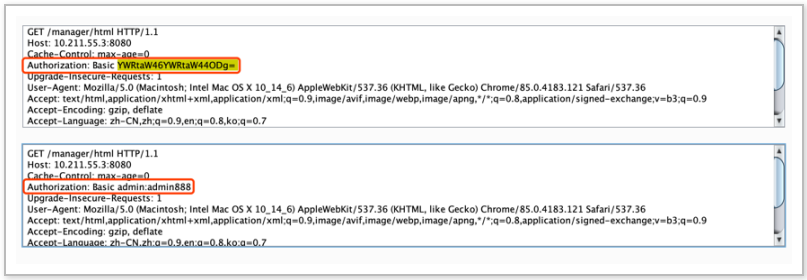
但是需要注意的是，这里需要在 tomcat 配置中开启列目录，将 false 改为 true，否则就会出现如下情形，无法利用该方式读取文件。

利用该方法进行文件读取的攻击方式现在已经几乎绝迹，也只能在一些内网中找到这样的古董系统，至于古董系统有没有开放 administration tool，就要看命了！

Manager 管理页面

Manager 管理平台我们都很熟悉，也是最常见的，包含多个管理模块，开启后方便开发及运维人员对 tomcat 项目发布进行管理。Manager 管理平台默认安装后是没有设置登录口令的，需要在 tomcat-user.xml 文件中进行配置，与上文的 admin 管理平台相同。

在登录 manager 后台时，tomcat 使用的是 Basic 认证方式，在请求的数据包中包含一个 Authorization 字段，该字段的值为账号密码的 base64 编码，如图所示：



Tomcat manager 包含 4 个不同的角色：

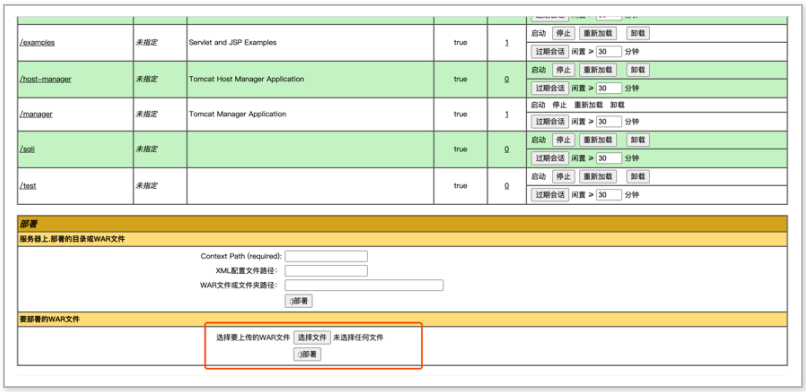
- manager-gui：允许访问 html 页面接口 (即 URL 路径为 / manager/html/*)
- manager-script：允许访问纯文本接口 (即 URL 路径为 / manager/text/*)

- manager-jmx：允许访问 JMX 代理接口 (即 URL 路径为 / manager/jmxproxy/*)
- manager-status：允许访问 Tomcat 只读状态页面 (即 URL 路径为 / manager/status/*)

其中 manager-gui、manager-script、manager-jmx 三个角色均具备 manager-status 角色的权限，即这三种角色权限无需再额外添加 manager-status 权限。实际使用中只需配置 manager-gui 角色通过 html 页面的形式访问管理平台。下面我们来分别简述一下 manager 的这 4 个角色。

manager-gui

manager-gui 是最常见也是最常用的模块，我们通常访问 / manager/html 看到的页面就是 manager-gui，不同版本之间的功能都大同小异，对于攻击者来说，最直接的方式就是通过部署应用的功能来部署 war 包，从而部署 webshell 后门应用。

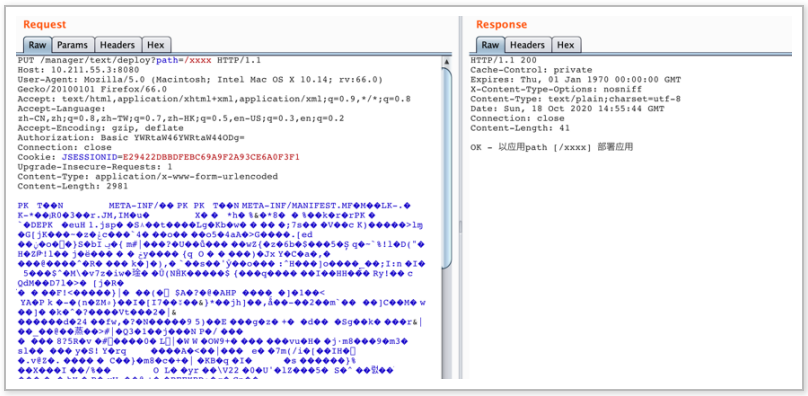


manager-script

该模块下包含了所有管理功能的接口，攻击者也可以通过这个接口来对 tomcat 应用发起攻击。下面我们来列举几个常用的功能：

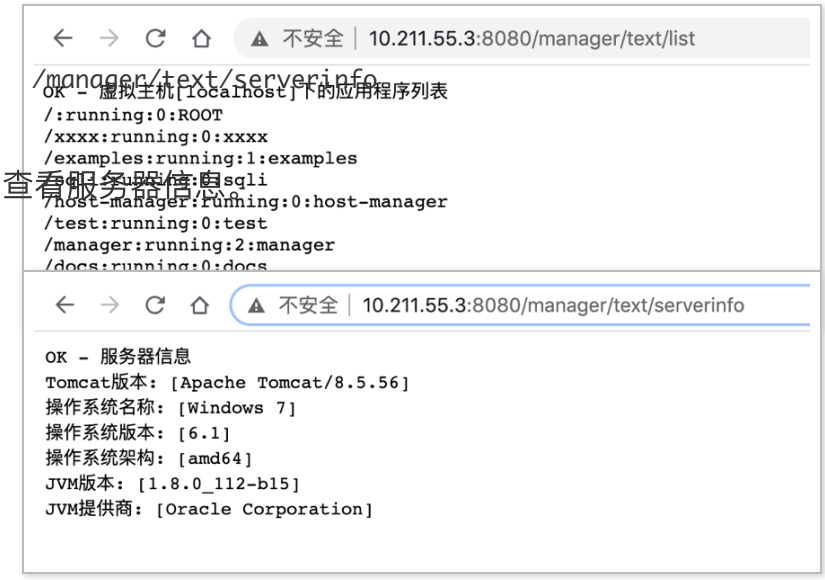
```
/manager/text/deploy?path=/xxx
```

部署 web 应用，需要注意的是这里部署的文件为 war 包，在请求时需要使用 PUT 方法。




```
/manager/text/list
```

查看所有部署的应用。



查看服务器信息

详细功能，可以在tomcat的帮助文档中查看：http://{ip}:{port}/

manager-status

该模块访问后可以看到一些与服务器相关的信息，没有太多有实际价值的信息，也只能帮助收集到一些简单的信息。

manager-jmx

Tomcat 使用 JMX 管理方式，在 Tomcat 的自带应用 manager 就是使用了 JMX 方式来管理 Tomcat，以此完成 Web 应用的动态部署、启动、停止。在 tomcat 的帮助文档中，提供了下面几种方式发送请求获取相应的信息：

```
query命令：http://[ip]:[port]/manager/jmxproxy/?qry=
get命令：http:// [ip]:[port]/manager/jmxproxy/?get=
set命令：http:// [ip]:[port]/manager/jmxproxy/?set=
invoke命令：http:// [ip]:[port]/manager/jmxproxy/?invok
```

访问上面的地址，我们就可以看到不同的信息，通过向不同的参数传递特定的参数，也可以获取到一些敏感信息。不加参数时查询到的是所有的 MBeans 的内容，加参数之后就可以查看到具体的 MBeans 的内容。

例如：

```
http://[ip]:[port]/manager/jmxproxy/?qry=*%3atype=User
```

该查询可以看到设置的 tomcat-user.xml 中配置的账号密码，甚至还可以通过 set 命令修改账号密码，来设置一个后门账号。

除此之外，tomcat 还可以开启 JMX Sorvice，默认情况下 Tomcat 是不开启 JMX service 的，开启 JMX Service 需要对 /bin 目录下的 Catalina.bat/Catalina.sh 文件进行更改。

Catalina.bat:

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote -Dcom
```

Catalina.sh:

```
CATALINA_OPTS="$CATALINA_OPTS-Dcom.sun.management.jmxr
```

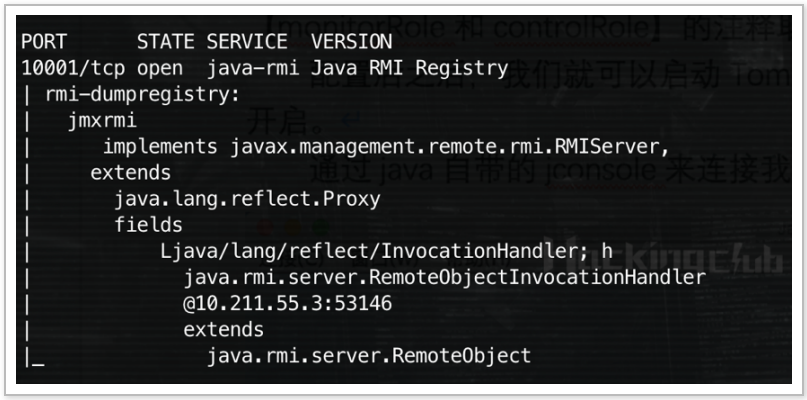
这里没有对 jmx 连接设置身份认证，从安全角度来说，需要设置身份认证，设置认证时，需要在配置中添加:

```
-Dcom.sun.management.jmxremote.password.file=path/jmxr
-Dcom.sun.management.jmxremote.access.file=path/jmxrem
```

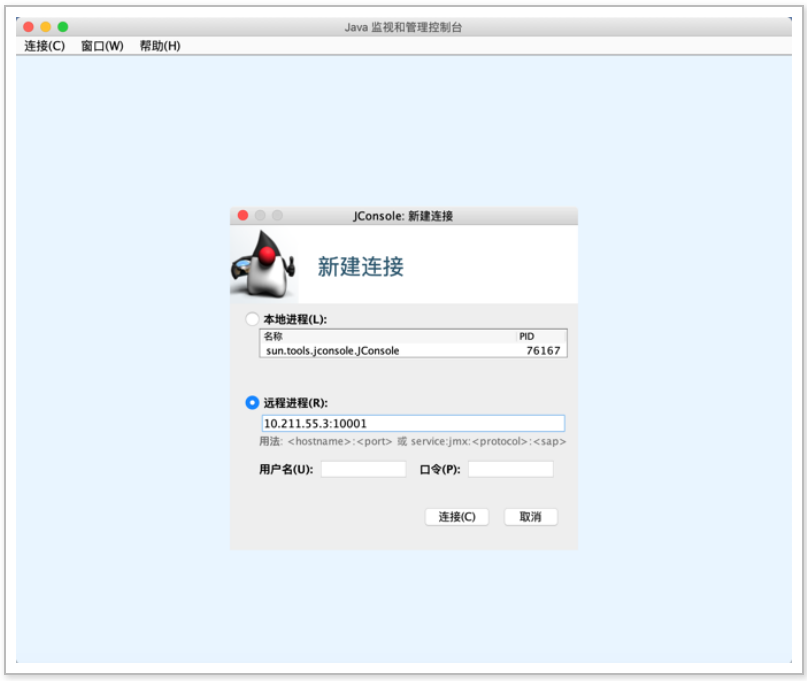
在 jdk 的安装目录 / jre/lib/management 目录下，有相应的 jmxremote.access 文件，将文件最后两行显示【monitorRole 和 controlRole】的注释取消，其中 monitorRole 为只拥有只读权限的角色，controlRole 有更高权限：读写等。默认情况下该目录下不存在 jmxremote.password 文件，我们可以将 jmxremote.password.template 文件改名，或者复制出来一份再改名即可，然后修改 jmxremote.password 文

件。同样将文件最后两行显示【monitorRole 和 controlRole】的注释取消。然后保存。

配置后之后，我们就可以启动 Tomcat，启动之后先查看一下我们配置的 jmx 是否正常开启。

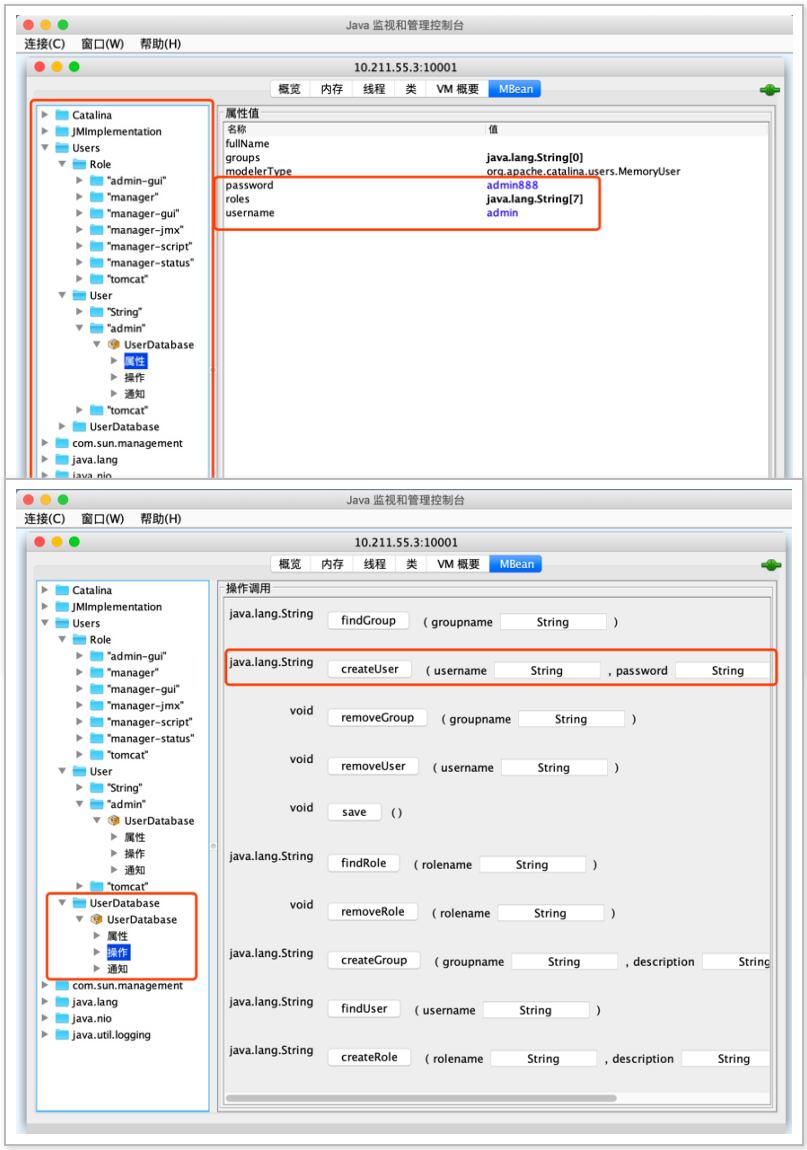


通过 java 自带的 jconsole 来连接我们配置 jmx 端口。

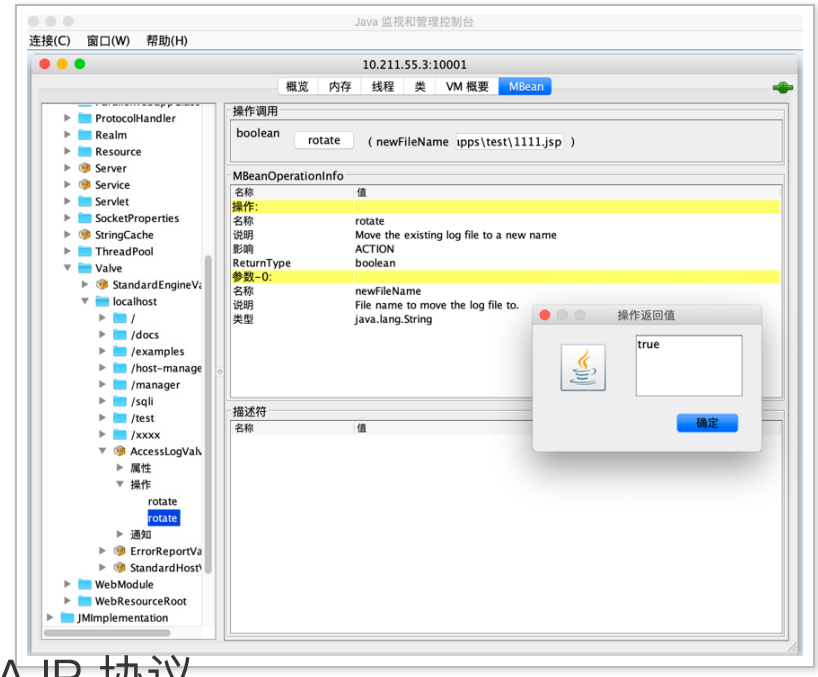


连接之后，我们可以通过 jconsole 的页面看到许多的敏感信息，这些与前面的 manager-jmx 类似。

同样的可以通过查看 MBean 看到 tomcat manager 配置的账号密码，除此之外，还能添加账号，从而添加后门账号。

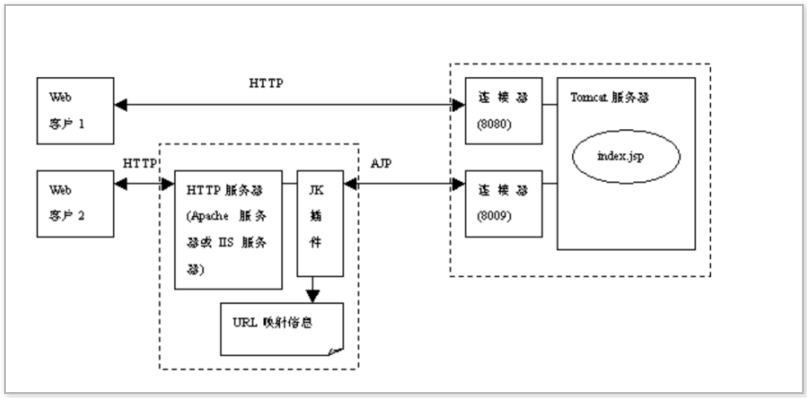


在 Catalina->Valve->localhost->AccessLogValve 中，可以实现对日志备份的操作，通过前面获取到的 Tomcat 物理路径，根据 webapps 目录以及从 MBean 中获取到的已安装的应用，可以构造出 web 路径。我们在进行日志备份时，需要备份到不存在的文件中，如果文件已存在，该功能不会对已存在的文件进行覆盖或追加。



AJP 协议

由于 tomcat 的 html 和图片解析功能相对其他服务器如 apche 等较弱，所以，一般都是集成起来使用，只有 jsp 和 servlet 服务交由 tomcat 处理，而 tomcat 和其他服务器的集成，就是通过 ajp 协议来完成的。Web 客户访问 Tomcat 服务器上 JSP 组件的两种方式如图所示。



在 tomcat 中有两个连接器，一个是监听在 8080 端口，负责建立 web 的 HTTP 连接，一个是监听在 8009 端口，负责 Tomcat 与其他 HTTP 服务器进行集成。配置文件 server.xml 中，关于 AJP 的配置项默认是关闭的，若需要开启，将注释符去掉，重新启动 Tomcat 即可。

相对而言，AJP 协议的攻击利用方式较少，目前爆出的可直接利用的就是 CVE-2020-1938（AJP 文件包含漏洞）。

Debug 模式

Tomcat 默认情况下，debug 模式是不开启的，如果需要开启 debug 时，要对 / bin 目录下的 Catalina.bat/Catalina.sh 文件进行修改： Windows 下在 Catalina.bat 中添加:

```
SET CATALINA_OPTS=-server -Xdebug -Xnoagent -Djava.com
```

Linux 下在 Catalina.sh 中对 jpda 相关的内容进行修改

修改完成后，在 startup.sh 文件中添加 jpda，如下所示：

全部修改完成后，我们就可以启动 tomcat 了，此时 debug 模式就正常启动了。我们通过 nmap 对 tomcat debug 端口进行探测，发现该端口 service 信息为 jdwp。

实际上 Tomcat 的 debug 模式也是调用的 jvm 的调试接口，正如配置文件中显示的那样，最终是通过调用 jdwp 来实现。

JPDA(Java Platform Debugger Architecture,Java 平台调试架构)，由 Java 虚拟机后端和调试平台前端组成，JPDA 为 Java 平台上的调试器定义了一个标准的体系结构。该体系结构包括 3 个主要组成部分：JVM TI（Java 虚拟机工具接口）、JDI（Java 调试连线协议）和 JDWP（Java 调试接口）。

在一些低版本的 jdk 中会存在漏洞，可以通过 jdwp 来执行系统命令。所以对 Tomcat 的安全配置时，要关闭 debug 模式。

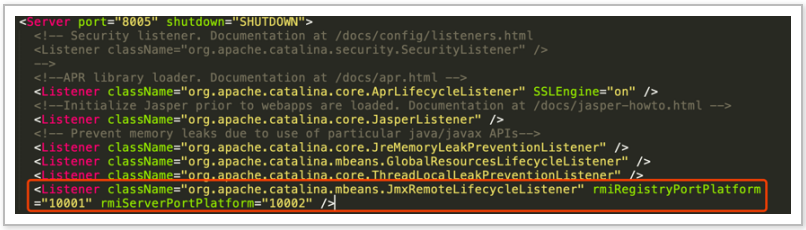
历史漏洞

CVE-2016-8735

漏洞说明:

该漏洞与之前 Oracle 发布的 mxRemoteLifecycleListener 反序列化漏洞（CVE-2016-3427）相关，是由于使用了 JmxRemoteLifecycleListener 的监听功能所导致。而在 Oracle 官方发布修复后，Tomcat 未能及时修复更新而导致的远程代码执行。

该漏洞所造成的最根本原因是 Tomcat 在配置 JMX 做监控时使用了 JmxRemoteLifecycleListener 的方法。



影响版本:

```
9.0.0.M1-9.0.0.M11
8.5.0-8.5.6
8.0.0.RC1-8.0.38
7.0.0-7.0.72
6.0.0-6.0.47
```

漏洞利用：

这里我们使用 ysoserial 来进行漏洞攻击利用

CVE-2017-12615 & CVE-2017-12616

漏洞说明：

该漏洞称之为 Tomcat PUT 方法任意写文件漏洞，类似 IIS 的 PUT 上传漏洞。该漏洞可以利用 HTTP 的 PUT 方法直接上传 webshell 到目标服务器，从而获取权限。该漏洞是高危漏洞，在 Tomcat 的 web.xml 默认情况下不存在该漏洞，但是一但开发者或者运维人员手动讲 web.xml 中的 readonly 设置为 false，可以通过 PUT / DELETE 进行文件操控。

CVE-2017-12616 是对 CVE-2017-12615 的绕过，影响 7.x、8.x、9.x 版本。

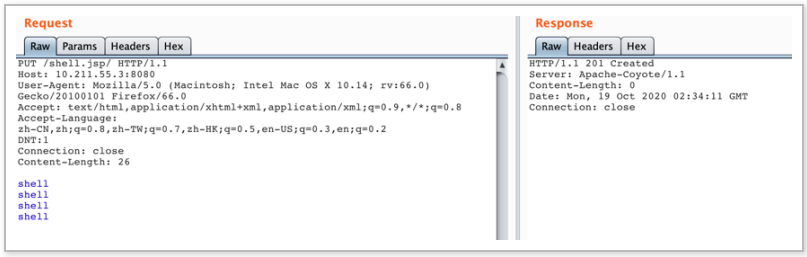
```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>listings</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>readonly</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

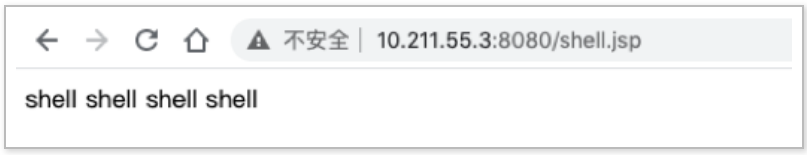
影响范围：

7.0.0 — 7.0.79

漏洞利用：

构造 PUT 上传数据包，上传时内容可以写 jsp 一句话木马，上传返回 201 时，表示上传成功，访问该文件即可。





上传绕过方法：

Windows：

> 1、利用 / shell.jsp::$DATA 的方式绕过
> 2、/shell.jsp%20，空格绕过 3、/shell.jsp/ ，Tomcat 在处理文件时会删除最后的 /

Linux：

> /shell.jsp/ ， Tomcat 在处理文件时会删除最后的 /

CVE-2019-0232

漏洞说明：

该漏洞是由于 Tomcat CGI 将命令行参数传递给 Windows 程序的方式存在错误，使得 CGIServlet 被命令注入影响。

该漏洞的利用条件较为苛刻，需同时满足下列条件：

1. 系统为 Windows
2. 启用了 CGI Servlet（默认为关闭）
3. 启用了 enableCmdLineArguments（Tomcat 9.0.* 及官方未来发布版本默认为关闭）

影响范围：

```
9.0.0.M1-9.0.17
8.5.0-8.5.39
7.0.0-7.0.93
```

漏洞利用：

CVE-2020-1938

漏洞描述：

漏洞是 Tomcat AJP 协议存在缺陷而导致，攻击者利用漏洞可以构造特定参数，读取服务器 webapp/ROOT 下的任意文件。若目标服务器同时存在文件上传功能，攻击者可进一步通过文件包含实现远程代码执行。

影响范围：

```
Apache Tomcat 6
Apache Tomcat 7 < 7.0.100
Apache Tomcat 8 < 8.5.51
Apache Tomcat 9 < 9.0.31
```

漏洞利用：

使用已公开的漏洞利用工具来进行漏洞利用。