云鼎实验室：Tomcat 远程代码执行漏洞分析（CVE-2017-12615）及补丁 Bypass

12小时之前
漏洞分析

作者：RicterZ@云鼎实验室

9 月 19 日，腾讯云安全中心监测到 Apache Tomcat 修复了2个严重级别的漏洞， 分别为： 信息泄露漏洞（CVE-2017-12616）、远程代码执行漏洞（CVE-2017-12615），在某些场景下，攻击者将分别能通过这两个漏洞，获取用户服务器上 JSP 文件的源代码，或是通过精心构造的攻击请求，向用户服务器上传恶意 JSP 文件，通过上传的 JSP 文件，可在用户服务器上执行任意代码。

云鼎实验室通过对于漏洞描述，搭建漏洞环境，并对其进行复现。此漏洞为高危漏洞，即使是非默认配置，但是一旦存在漏洞，那么攻击者可以成功上传 Webshell，并控制服务器。

复现

根据描述，在 Windows 服务器下，将 readonly 参数设置为 false 时，即可通过 PUT 方式创建一个 JSP 文件，并可以执行任意代码。

通过阅读 conf/web.xml 文件，可以发现：

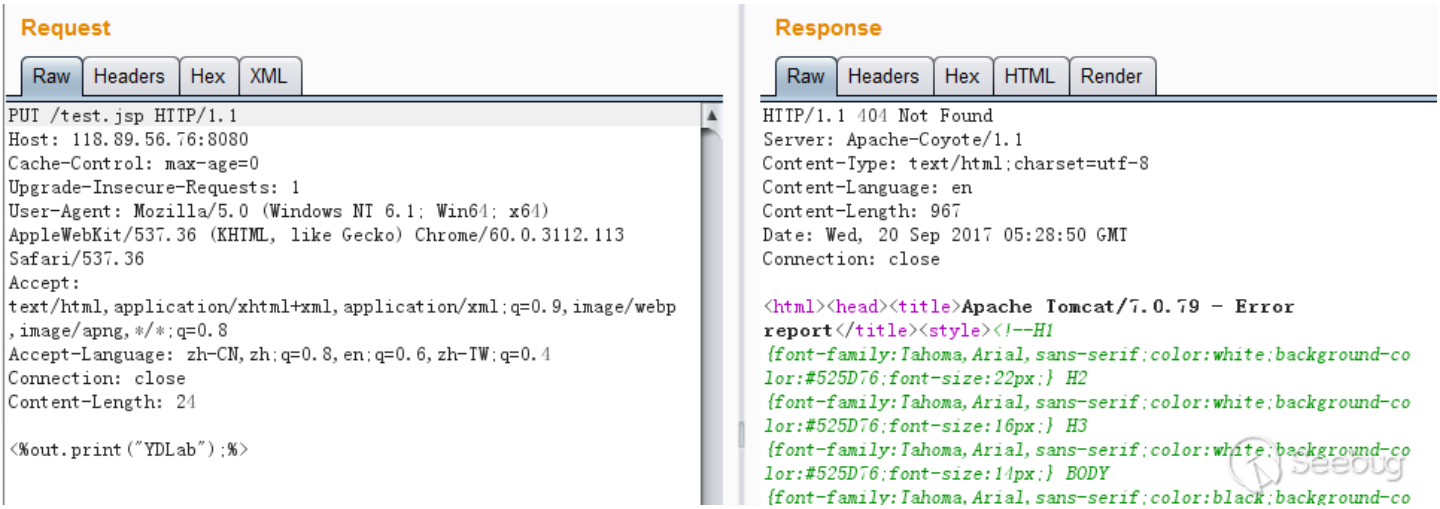
```
<!--   readonly     Is this context "read only", so HTTP          -->
<!--                commands like PUT and DELETE are              -->
<!--                rejected?  [true]                             -->
```


默认 readonly 为 true，当 readonly 设置为 false 时，可以通过 PUT / DELETE 进行文件操控。

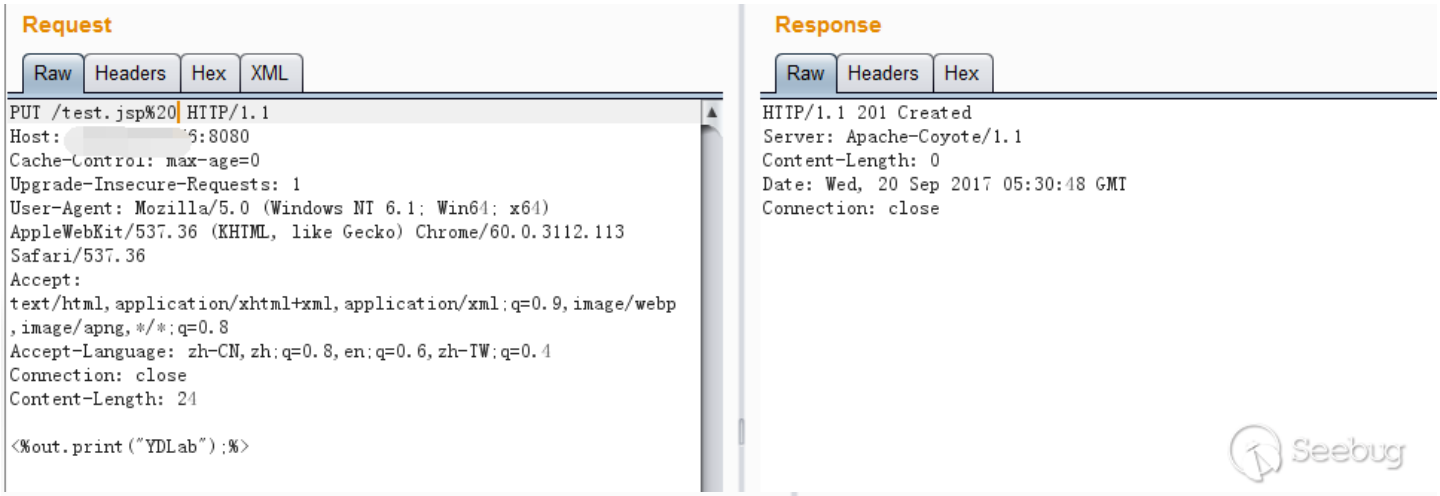
配置 readonly 为 false：

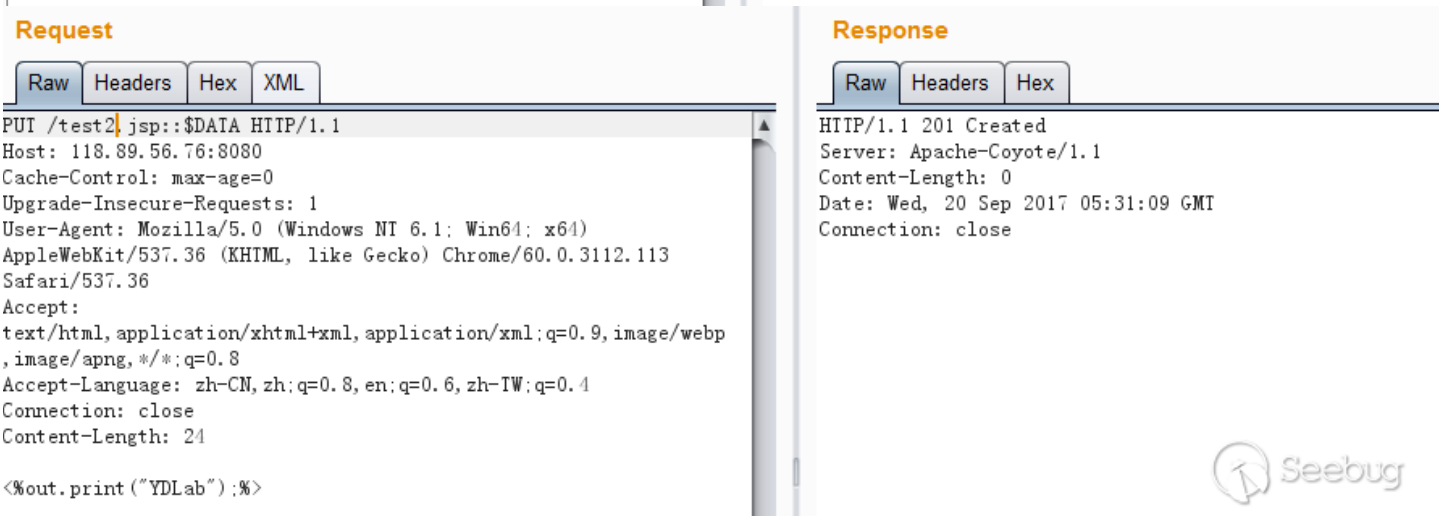
```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</serv
let-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>listings</param-name>
        <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>
```


启动 Tomcat，利用 PUT 请求创建文件：

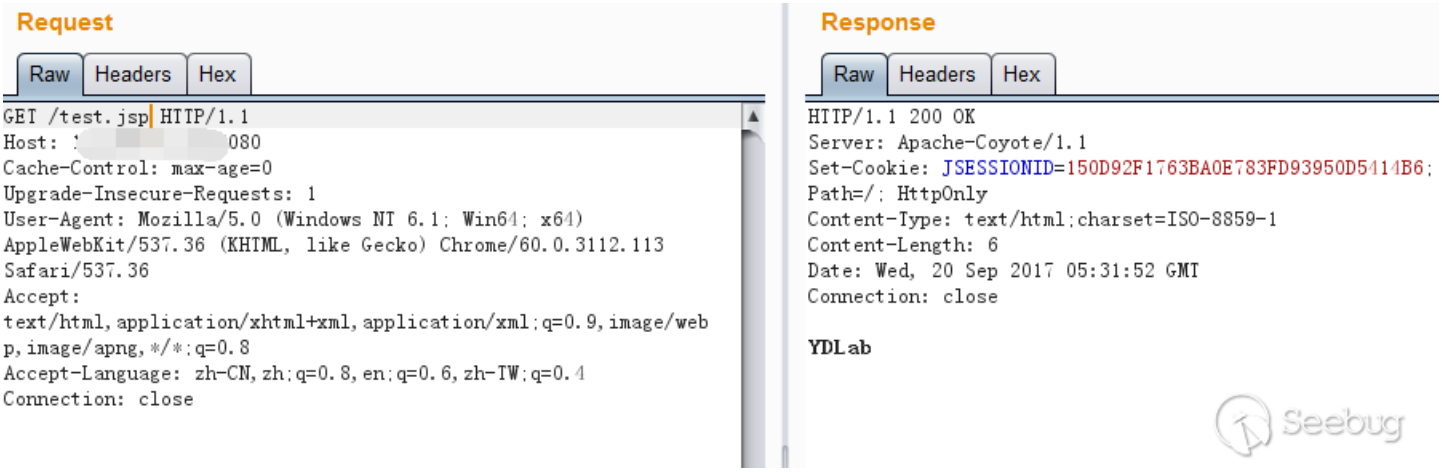


提示 404。通过描述中的 Windows 受影响，可以结合 Windows 的特性。其一是 NTFS 文件流，其二是文件名的相关限制（如 Windows 中文件名不能以空格结尾）来绕过限制：





访问发现可以正常输出：



分析

Tomcat 的 Servlet 是在 conf/web.xml 配置的，通过配置文件可知，当后缀名为 .jsp 和 .jspx 的时候，是通过 JspServlet 处理请求的：

```
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.jspx</url-pattern>
</servlet-mapping>
```


而其他的静态文件是通过 DefaultServlet 处理的：

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```


可以得知，“1.jsp ”（末尾有一个和空格）并不能匹配到 JspServlet，而是会交由 DefaultServlet 去处理。当处理 PUT 请求时：

```
protected void doPut(HttpServletRequest req, HttpServletResponse res
p)
    throws ServletException, IOException {

    if (readOnly) {
        resp.sendError(HttpServletResponse.SC_FORBIDDEN);
        return;
    }

    String path = getRelativePath(req);

...

    try {
        Resource newResource = new Resource(resourceInputStream);
        // FIXME: Add attributes
        if (exists) {
            resources.rebind(path, newResource);
        } else {
            resources.bind(path, newResource);
        }
    } catch(NamingException e) {
        result = false;
    }

...

}
```


会调用 `resources.bind` ：

```
public void bind(String name, Object obj)
    throws NamingException {
    dirContext.bind(parseName(name), obj);
    cacheUnload(name);
}
```

dirContext 为 FileDirContext：

```
@Override
public void bind(String name, Object obj, Attributes attrs)
    throws NamingException {

    // Note: No custom attributes allowed

    File file = new File(base, name);
    if (file.exists())
        throw new NameAlreadyBoundException
            (sm.getString("resources.alreadyBound", name));

    rebind(name, obj, attrs);

}
```


调用 rebind 创建文件：

```
public void rebind(String name, Object obj, Attributes attrs)
    throws NamingException {
...

    FileOutputStream os = null;
    byte buffer[] = new byte[BUFFER_SIZE];
    int len = -1;
    try {
        os = new FileOutputStream(file);
        while (true) {
            len = is.read(buffer);
            if (len == -1)
                break;
            os.write(buffer, 0, len);
        }
```


又由于 Windows 不允许“ ”作为文件名结尾，所以会创建一个 `.jsp` 文件，导致代码执行。

Bypass 分析

然而，经过黑盒测试，当 PUT 地址为/1.jsp/时，仍然会创建 JSP，会影响 Linux 和 Windows 服务器，并且 Bypass 了之前的补丁，分析如下。

在进入 bind 函数时，会声明一个 File 变量：

```
@Override
public void bind(String name, Object obj, Attributes attrs)
    throws NamingException {

    // Note: No custom attributes allowed

    File file = new File(base, name);
```

进入 File 后，会对 name 进行 normalize

```
public File(File parent, String child) {
    if (child == null) {
        throw new NullPointerException();
    }
    if (parent != null) {
        if (parent.path.equals("")) {
            this.path = fs.resolve(fs.getDefaultParent(),
                                   fs.normalize(child));
        } else {
            this.path = fs.resolve(parent.path,
                                   fs.normalize(child)); // 这里
```

最后得到的 path 就是没有最后 / 的 path 了: