RELIABLE NETWORK
SAFE WORLD
TOPSEC

Struts2 历史重大漏洞

分析与总结报告

天融信阿尔法实验室

目录

一 前言

2018 年，同往年一样，还是爆发了很多重大漏洞，如 Drupal 远程代码执行漏洞、S2-057 命令执行、Adobe Flash Player 任意代码执行等漏洞，其中包括了常年必有的 Struts2 RCE 漏洞。

Struts2 作为世界上最流行的 Java Web 服务器框架之一，全球很多行业包括但不限于金融、教育、医疗的系统都使用它当作底层框架，分布范围极广，可见其漏洞危害有多大，而 Struts2 漏洞却常年爆发。

为了让大家更加了解和重视 Struts2 漏洞，天融信阿尔法实验室编制了本报告，报告从 Struts2 的背景介绍、漏洞统计、漏洞数据、以及漏洞建议等方面对其进行了全面的阐述。

二 Struts2 背景介绍

2.1 Struts2 框架背景

Struts 是 Apache 基金会 Jakarta 项目组的一个开源项目，它采用 MVC 模式，能够很好地帮助 java 开发者利用 J2EE 开发 Web 应用。和其他的 java 架构一样，Struts 也是基于面向对象思想设计，其将 MVC 模式"分离显示逻辑和业务逻辑"的能力发挥得淋漓尽致。

2000 年 5 月发展至今，Struts 已经成为了一个高度成熟的框架，不管是稳定性还是可靠性都得到了广泛的证明。市场占有率超过 20%，拥有丰富的开发人群，几乎已经成为了事实上的工业标准。但是随着时间的流逝，技术的进步，Struts1 的局限性也越来越多地暴露出来，并且制约了 Struts1 的继续发展。

对于 Struts1 框架而言，由于与 JSP/Servlet 耦合非常紧密，因而导致了一些严重的问题。首先，Struts1 支持的表现层技术单一。由于 Struts1 出现的年代比较早，那个时候没有 FreeMarker、Velocity 等技术，因此它不可能与这些视图层的模版技术进行整合。其次，Struts1 与 Servlet API 的严重耦合，使应用难于测试。最后，Struts1 代码严重依赖于 Struts1 API，属于侵入性框架。

从技术层面上看，当时出现了许多与 Struts1 竞争的视图层框架，比如 JSF、Tapestry 和 spring MVC 等。这些框架由于出现的年代比较近，应用了最新的设计理念，同时也从 Struts1 中吸取了经验，克服了很多不足。这些框架的出现也促进了 Struts 的发展。借此 Struts1 已经分化成了两个框架：第一个是在传统的 Struts1 的基础上，融合了另外的一个优秀的 Web 框架 WebWork 的 Struts2。Struts 2 虽然是在 Struts1 的基础上发展起来的，但是实质上是以 WebWork 为核心的。

作为 Struts 的下一代产品，Struts2 与 Struts1 的体系结构差别巨大。Struts2 为传统的 Struts1 注入了 WebWork 的先进的设计理念，统一了 Struts1 和 WebWork 两个框架。它是一个简洁的，可扩展的基于 MVC 设计模式的框架，用于创建企业准备的 Java Web 应用程序。设计这个框架是为了从程序的构建、部署、应用等方面来简化整个开发周期。

2.2 Struts2 漏洞简述

自编号 S2-001 的 Struts2 命令执行漏洞被发现到至今的编号 S2-057。漏洞总数已经上升至 57 个之多，时间上横跨十余载。漏洞的种类更是繁多：

编号 S2003、S2005、S2007、S2-009、S2-012、S2-013、S2-015、S2-016、S2-019、S2-029、S2-032、S2-033、S2-037 等命令执行类漏洞，利用了 Struts2

框架执行了用户传进来的恶意 OGNL 表达式造成代码执行。注入点包含了 request 的参数名、cookie 名、参数值、filename、URL、content-type。

编号 S2-002、S2006、S2-028 等过滤不严格造成的 XSS 漏洞，如<s：url>和<s：a>标签中注入未正确的转义参数，XWork 默认未过滤 action 来进行 XSS 攻击。编号 S2004 目录遍历类漏洞，允许攻击者遍历目录结构并使用双重编码的 URL 和相对路径下载“静态”内容文件夹之外的文件。

编号 S2010、S2023、S2038 等安全 bypass 类漏洞，包含滥用已知的会话属性来绕过令牌检查，令牌的生成值可以预测，甚至可以绕过令牌验证来进行 CSRF 攻击编号 S2017 重定向类漏洞，通过操作前缀“redirect：”/“redirectAction：”的参数引入的漏洞允许打开重定向。

编号 S2011、S2051、S2054、S2056 等 DOS 类攻击漏洞，包含了频繁请求构造的超长参数，频繁执行利用 REST 插件使用的 JSON-LIB、XStream 库构造带有特质的有效负载的请求。当然还有一些未分类的漏洞这里不一一举例了。漏洞的危害等级从低危到高危都有所涉及。

2.3 Struts2 影响概述

由于 Struts2 框架的普及性，漏洞一经爆发，影响范围非常广泛，众多行业遭受波及，包括但不限于教育，政府，金融，互联网，通信行业。北京，上海，广州，沿海城市等经济发达地区沦为漏洞高发区。

Struts2 框架的漏洞一直是高校网络中存在的安全顽疾，由于学校中有很多信息系统在开发阶段都使用了 Struts2 作为底层框架，而后期运行的人员并不清楚底层架构从而无法判断漏洞是否存在，如果开发人员离职或是停止技术支持，那漏洞就可能长

期存在，并持续造成危害。黑客可以直接利用该漏洞通过浏览器在远程服务器上执行任意系统命令，将会对受影响的学校站点造成严重影响，引发数据泄露、网页篡改、植入后门、成为肉鸡等安全事件。

2.4 Struts2 修复情况

至今为止，各大安全厂商和安全组织已把此系列漏洞作为重大高危漏洞进行处理，一经发现，会第一时间进行分析预警，并给出解决方案，并在后续进行详细的漏洞分析。针对于今年 8 月 22 日，编号 S2-057 漏洞出现，天融信阿尔法实验室在第一时间提供了此漏洞的详细信息和漏洞描述，影响的版本范围，漏洞的解决方案和厂商的补丁。企业和个人可针对自己的安全策略和流程可进行快速修复。

三 Struts2 漏洞统计及分析

3.1 Struts2 历史全版本漏洞简介

从 07 年第一个 Struts2 漏洞 S2001 至今，编号已经到了 S2057，漏洞个数已经达到了 57 个，据天融信阿尔法实验室分析，这些漏洞包含了 RCE、XSS、CSRF、DOS、目录遍历和其他功能缺陷漏洞等多种类型，其中命令执行多达 17 个。

当然这些漏洞并不是所有都被人们所关注，有一些危害不大的并不为人所知，当然，也有一些漏洞在当年风靡一时如：S2-003、S2-005、S2-007、S2-008、S2-009、S2-012、S2-013、S2-015、S2-016、S2-019、S2-029、S2-032、S2-033、S2-037、S2-045、S2-046、S2-048、S2-052、S2-055、S2-057。历经十多个年头 Struts2 每年总会报出新的漏洞，只有对这些漏洞进行分析，才能更好的防御，才能在新的漏洞出现的时候做到提前预防。

3.2 Struts2 重大漏洞分析

3.2.1 S2-032

漏洞描述

当启用动态方法调用时，攻击者可以构造恶意表达式，在服务器端执行任意代码，获取服务器权限。

受影响的系统版本

Struts 2.3.20 - Struts 2.3.28（2.3.20.3 和 2.3.24.3 除外）

漏洞编号

CVE-2016-3081

漏洞分析

漏洞存在于 Struts2 的动态方法引用功能。只要在 Struts2 配置文件中开启该功能，就可能被利用。

<constant name="struts.enable.DynamicMethodInvocation" value="true" />

如果我们请求 http：//localhost/index.action?method:OGNL 的情况下，请求的 OGNL 表达式会被执行，造成命令执行。

method 后面跟的方法名会被 Struts2 进行解析，代码位于 DefaultActionMapper.java 中。

```
131     protected Container container;
132
133     public DefaultActionMapper() {
134         prefixTrie = new PrefixTrie() {
135             {
136                 put(METHOD_PREFIX, new ParameterAction() {
137                     public void execute(String key, ActionMapping mapping) {
138                         if (allowDynamicMethodCalls) {
139                             mapping.setMethod(key.substring(METHOD_PREFIX.length()));
140                         }
141                     }
142                 });
143
144                 put(ACTION_PREFIX, new ParameterAction() {
145                     public void execute(final String key, ActionMapping mapping) {
146                         if (allowActionPrefix) {
147                             String name = key.substring(ACTION_PREFIX.length());
148                             if (allowDynamicMethodCalls) {
149                                 int bang = name.indexOf('!');
150                                 if (bang != -1) {
151                                     String method = name.substring(bang + 1);
152                                     mapping.setMethod(method);
```

可以看到 mapping.setMethod(key.substring(METHOD_PREFIX.length()));将我们传入的方法加入到 map 中。然后在 DefaultActionInvocation.java 中，被 invokeAction 引用。

```
417     protected String invokeAction(Object action, ActionConfig actionConfig) throws Exceptio>
418         String methodName = proxy.getMethod();
419
420         if (LOG.isDebugEnabled()) {
421             LOG.debug("Executing action method = #0", methodName);
422         }
423
424         String timerKey = "invokeAction: " + proxy.getActionName();
425         try {
426             UtilTimerStack.push(timerKey);
427
428             Object methodResult;
429             try {
430                 methodResult = ognlUtil.getValue(methodName + "()", getStack().getContext()>
431             } catch (MethodFailedException e) {
432                 // if reason is missing method, try find version with "do" prefix
433                 if (e.getReason() instanceof NoSuchMethodException) {
434                     try {
435                         String altMethodName = "do" + methodName.substring(0, 1).toUpperCas>
436                         methodResult = ognlUtil.getValue(altMethodName, getStack().getConte>
437                     } catch (MethodFailedException e1) {
438                         // if still method doesn't exist, try checking UnknownHandlers
```

invokeAction 首先获取方法字符串，然后调用 ognlUtil.getValue 来执行方法并获取方法结果。问题就出在传给 ognl 的 methodName 没有进行严格的过滤，尤其是没有过滤 ognl 关键字，从而造成命令执行。

3.2.2 S2-037

漏洞描述

漏洞是由于对 method 没有进行过滤导致的，使用到 REST 插件的 Struts2 应用，会被攻击者实现远程代码执行攻击。

受影响的系统版本

Struts 2.3.20 – Struts 2.3.28.1

漏洞编号

CVE-2016-4438

漏洞分析

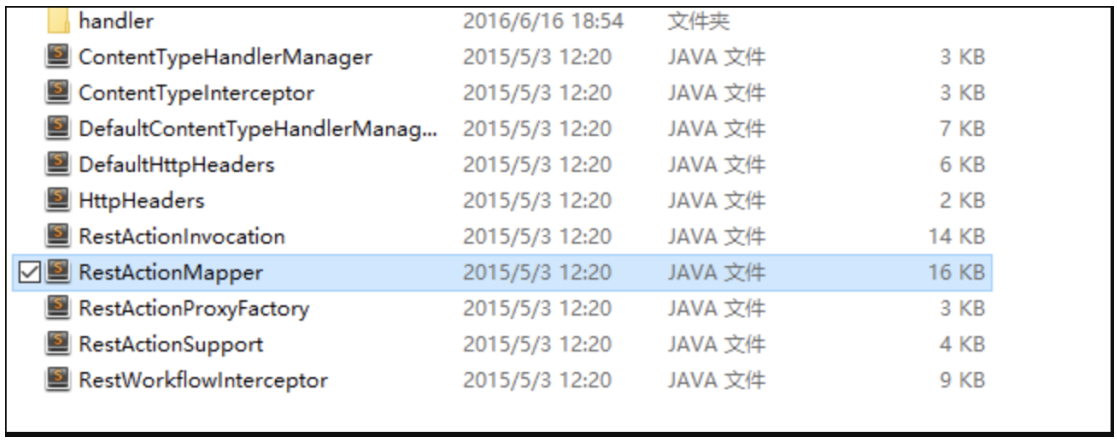
S2037 是基于 033 的一个绕过，在 033 中，需要开启动态方法执行，也就是 032 的条件，同时还需要安装 rest 插件。而 037 中，不需要开启动态方法执行就能触发代码执行漏洞。

首先看看 033 触发过程

开启动态方法执行需要在 struts.xml 中配置，这和 032 没区别

```
<constant name="struts.enable.DynamicMethodInvocation" value="true" />
```

033 问题出现在 rest 插件中 RestActionMapper.java 中



在 getMapping 方法中，在处理动态代码执行过程中设置的属性没有做过滤

```
    // handle "name!method" convention.
    handleDynamicMethodInvocation(mapping, mapping.getName());

private void handleDynamicMethodInvocation(ActionMapping mapping, String name) {
    int exclamation = name.lastIndexOf("!");
    if (exclamation != -1) {
        mapping.setName(name.substring(0, exclamation));
        if (allowDynamicMethodCalls) {
            mapping.setMethod(name.substring(exclamation + 1));
        } else {
            mapping.setMethod(null);
        }
    }
}
```

因为在 uri 中的！后面的属性加入到了 mapping 中，然后如果开启了动态代码执行，也就是 allowDynamicMethodCalls 开启，最后 method 会进入到 com.opensymphony.xwork2.DefaultActionInvocation 类的 invokeAction 方法，之后的就是跟 032 一样了，可以参考之前的我们分析 Struts2 漏洞的文章。

接下来看看 037 怎么绕过开启动态方法，通杀所有 rest 插件的。

```
// handle "name!method" convention.
handleDynamicMethodInvocation(mapping, mapping.getName());

String fullName = mapping.getName();
// Only try something if the action name is specified
if (fullName != null && fullName.length() > 0) {

    // cut off any ;jsessionid= type appendix but allow the rails-like ;edit
    int scPos = fullName.indexOf(';');
    if (scPos > -1 && !"edit".equals(fullName.substring(scPos + 1))) {
        fullName = fullName.substring(0, scPos);
    }

    int lastSlashPos = fullName.lastIndexOf('/');
    String id = null;
    if (lastSlashPos > -1) {

        // fun trickery to parse 'actionName/id/methodName' in the case of 'animals/dog/edit'
        int prevSlashPos = fullName.lastIndexOf('/', lastSlashPos - 1);
        if (prevSlashPos > -1) {
            mapping.setMethod(fullName.substring(lastSlashPos + 1));
            fullName = fullName.substring(0, lastSlashPos);
            lastSlashPos = prevSlashPos;
        }
        id = fullName.substring(lastSlashPos + 1);
    }
```

在调用完 handleDynamicMethodInvocation(mapping, mapping.getName());后面又调用了

```
mapping.setMethod(fullName.substring(lastSlashPos + 1));
```

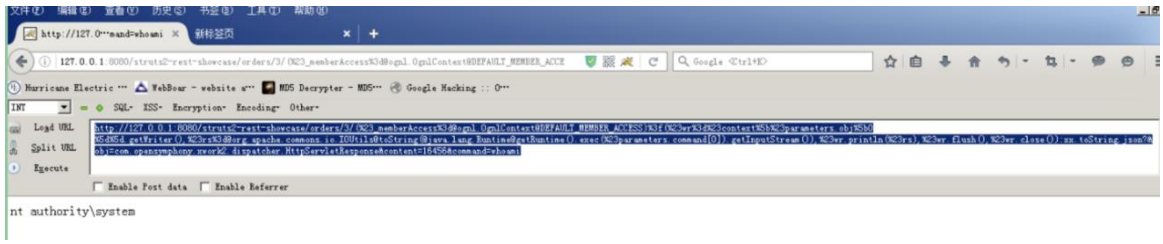
同时看到这里没有判断

```
if (allowDynamicMethodCalls) {
```

所以这个地方不需要开启动态执行，同时也在这里给出了提示

```
// fun trickery to parse 'actionName/id/methodName' in the case of 'animals/dog/edit'
```

这里加上了三目运算符才绕过测试结果如下：



可以看到成功执行了 system 命令。

3.2.3 S2-048

漏洞描述

在 Struts 2.3.X 系列的 Showcase 插件中演示 Struts2 整合 Struts 1 的插件中存在一处任意代码执行漏洞。当你的 Web 应用使用了 Struts 2 Struts 1 插件，则可能导致 Struts2 执行由外部输入的恶意攻击代码。

受影响的系统版本

Struts 2.3.x 系列中启用了 struts2-struts1-plugin 插件的版本。

漏洞编号

CVE-2017-9791

漏洞分析

漏洞出在 org.apache.struts2.s1.Struts1Action 类的 execute 方法中。

```
@Inject
public void setConfiguration(Configuration config) {
    this.configuration = config;
}

public String execute() throws Exception {
    ActionContext ctx = ActionContext.getContext();
    ActionConfig actionConfig = ctx.getActionInvocation().getProxy().getConfig();
    Action action = null;
    try {
        action = (Action) objectFactory.buildBean(className, null);
    } catch (Exception e) {
        throw new StrutsException("Unable to create the legacy Struts Action", e, actionConfig);
    }

    // We should call setServlet() here, but let's stub that out later

    Struts1Factory strutsFactory = new Struts1Factory(Dispatcher.getInstance().getConfigurationManager().getConfiguration());
    ActionMapping mapping = strutsFactory.createActionMapping(actionConfig);
    HttpServletRequest request = ServletActionContext.getRequest();
    HttpServletResponse response = ServletActionContext.getResponse();
    ActionForward forward = action.execute(mapping, actionForm, request, response);

    ActionMessages messages = (ActionMessages) request.getAttribute(Globals.MESSAGE_KEY);
    if (messages != null) {
        for (Iterator i = messages.get(); i.hasNext(); ) {
            ActionMessage msg = (ActionMessage) i.next();
            if (msg.getValues() != null && msg.getValues().length > 0) {
                addActionMessage(getText(msg.getKey(), Arrays.asList(msg.getValues())));
            } else {
                addActionMessage(getText(msg.getKey()));
            }
        }
    }
```

首先这里 ActionForward forward = action.execute(mapping, actionForm, request, response)，会把表单的请求处理回显给客户端。

然后 addActionMessage(getText(msg.getKey(), Arrays.asList(msg.getValues())));这里又调用了 getText 方法会把 action messages 传递给 com.opensymphony.xwork2.util.LocalizedTextUtil.getDefaultMessage ，最后在其中又调用了 com.opensymphony.xwork2.util.TextParseUtil.translateVariables，它用来调用 OGNL 表达式用处理字符串表达式，利用 OGNL 表达式执行任意命令。

3.2.4 S2-052

漏洞描述

Struts2 REST 插件的 XStream 组件存在反序列化漏洞，使用带有 XStream 实例的 XStreamHandler 进行反序列化操作时，未对数据内容进行有效验证，存在安全隐患，可被远程攻击。

受影响的系统版本

Struts 2.5 – Struts 2.5.12

漏洞编号

CVE-2017-9805

漏洞分析

通过发送 payload 请求后，返回报错信息，可以看到关键的调用栈位置，

在 ContentTypeInterceptor.intercept 之后，再将 reader 中的数据通过

XStreamHandler 中的 toObject 方法进行实例化，

```
public class ContentTypeInterceptor extends AbstractInterceptor
{
  private static final long serialVersionUID = 1L;
  ContentTypeHandlerManager selector;

  @Inject
  public void setContentTypeHandlerSelector(ContentTypeHandlerManager sel)
  {
    this.selector = sel;
  }

  public String intercept(ActionInvocation invocation) throws Exception {
    HttpServletRequest request = ServletActionContext.getRequest();
    ContentTypeHandler handler = this.selector.getHandlerForRequest(request);

    Object target = invocation.getAction();
    if ((target instanceof ModelDriven)) {
      target = ((ModelDriven)target).getModel();
    }

    if (request.getContentLength() > 0) {
      InputStream is = request.getInputStream();
      InputStreamReader reader = new InputStreamReader(is);
      handler.toObject(reader, target);
    }
    return invocation.invoke();
  }
}
```

然后接着调用了 poayload 中的 unmarshaller 对 xml 数据进行反序列化，

```
<map>
<entry>
<jdk.nashorn.internal.objects.NativeString>
<value class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data">
<dataHandler>
<dataSource class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource">
<is class="javax.crypto.CipherInputStream">
<cipher class="javax.crypto.NullCipher">
```

最终执行了 payload 中 java.lang.ProcessBuilder 进程生成器中的 command 方法来执行系统程序和参数。

```
<next class="java.lang.ProcessBuilder">
<command>
<string>c:\windows\system32\cmd.exe</string>
<string>/c echo topsec> c:/topsec.txt</string>
</command>
```

3.2.5 S2-057

漏洞描述

当 struts.mapper.alwaysSelectFullNamespace 设置为 true，并且 package 标签页以及 result 的 param 标签页的 namespace 值的缺失，或使用了通配符时可造成 namespace 被控制，最终 namespace 会被带入 OGNL 语句执行，从而产生远程代码执行漏洞。

受影响的系统版本

Struts 2.3 - Struts 2.3.34

Struts 2.5 - Struts 2.5.16

漏洞编号

CVE-2018-11776

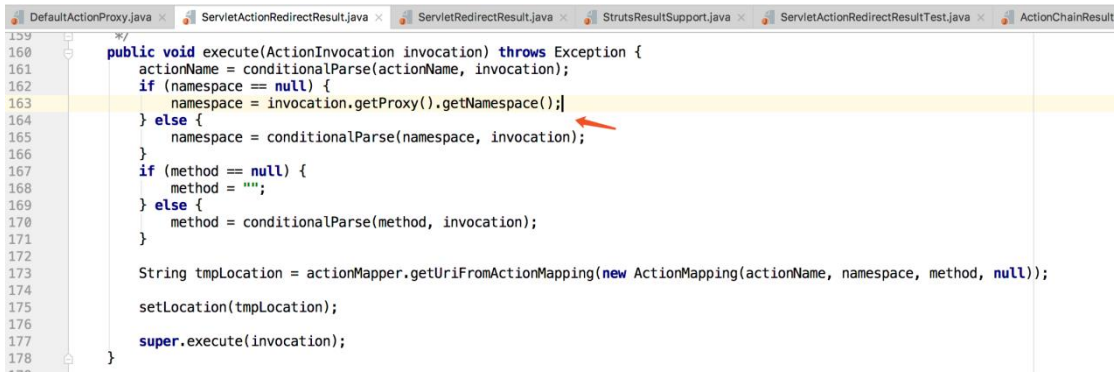
漏洞分析

在 DefaultActionMapper 这个类的 parseNameAndNamespace 方法里。

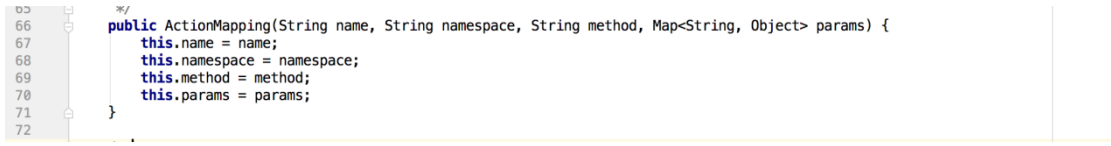
```
342    protected void parseNameAndNamespace(String uri, ActionMapping mapping, ConfigurationManager configManager) {
343        String namespace, name;
344        int lastSlash = uri.lastIndexOf('/');
345        if (lastSlash == -1) {
346            namespace = "";
347            name = uri;
348        } else if (lastSlash == 0) {
349            // ww-1046, assume it is the root namespace, it will fallback to
350            // default
351            // namespace anyway if not found in root namespace.
352            namespace = "/";
353            name = uri.substring(lastSlash + 1);
354        } else if (alwaysSelectFullNamespace) {
355            // Simply select the namespace as everything before the last slash
356            namespace = uri.substring(0, lastSlash);
357            name = uri.substring(lastSlash + 1);
358        } else {
359            // Try to find the namespace in those defined, defaulting to ""
360            Configuration config = configManager.getConfiguration();
361            String prefix = uri.substring(0, lastSlash);
362            namespace = "";
363            boolean rootAvailable = false;
364            // Find the longest matching namespace, defaulting to the default
365            for (PackageConfig cfg : config.getPackageConfigs().values()) {
366                String ns = cfg.getNamespace();
367                if (ns != null && prefix.startsWith(ns) && (prefix.length() == ns.length() || prefix.charAt(ns.length()) == '/')) {
368                    if (ns.length() > namespace.length()) {
369                        namespace = ns;
370                    }
371                }
372                if ("/".equals(ns)) {
373                    rootAvailable = true;
374                }
375            }
376
377            name = uri.substring(namespace.length() + 1);
```

当 alwaysSelectFullNamespace 被设置为 true 时，namespace 的值是从 URL 中获取的。URL 是可控的，所以 namespace 也是可控的。

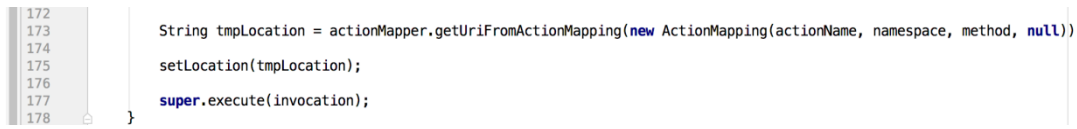
Action 执行结束之后，程序会调用 ServletActionRedirectResult 类中的 execute()方法进行重定向 Result 的解析。



首先，当 namespace 为空时，调用 invocation.getProxy().getNamespace()赋值给变量 namespace，然后将变量 namespace 传入 ActionMapping 构造函数中。



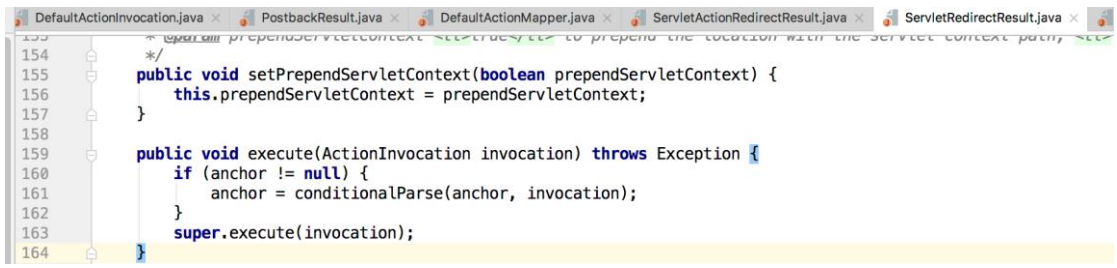
然后，ActionMapper.getUriFromActionMapping()对 ActionMapping 后的值进行重组，生成一个 URL 字符串（包含 namespace），并赋值给了 tmplocation 变量。



紧接着将带有 namespace 的 tmplocation 传入了 setLocation()方法中。

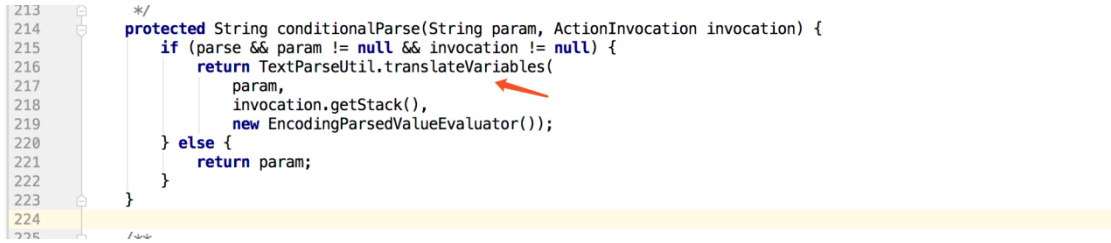
该方法将 tmpLocation 值赋值给了 StrutsResultSupport 类中的 location 变量。

然后，跟踪 super.execute()方法。



继续跟踪 ServletActionResult 类中的 super.execute()。

在 StrutsResultSupport 类中的 execute()方法中，刚刚被赋值的 location 变量（带有 namespace 的）被传入了 conditionalParse()方法。



最终，通过 TextParseUtil.translateVariables()对 namespace 进行 OGNL 解析，导致远程代码执行漏洞。

四 Struts2 漏洞数据分析

4.1 Struts2 漏洞历史时间统计

Struts2 发布于 2007 年，如图 3-1 所示，2007 年发布当年就爆发了一个漏洞，随后漏洞数量每年呈现递增的趋势。到 2016 年，漏洞数量达到 18 个。2017 年漏洞数量稍有下降，但也同样保持在 2 位数的水平。

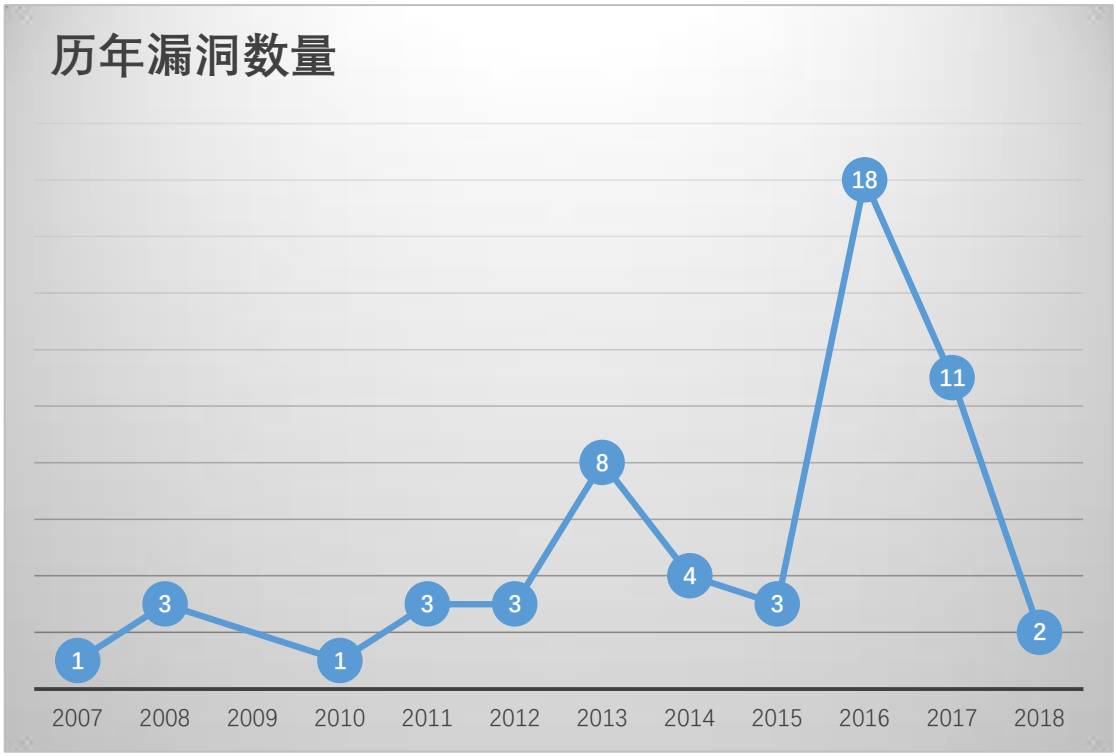


图 4-1 历史漏洞统计

4.2 Struts2 漏洞类型统计

从图 3-2 可以看出，在近 10 年来爆发的 57 个 Struts2 漏洞中，命令执行类漏洞有 28 个，占总量的 49%。占据第二位的是 DOS 类漏洞，有 10 个，占比为 17%。

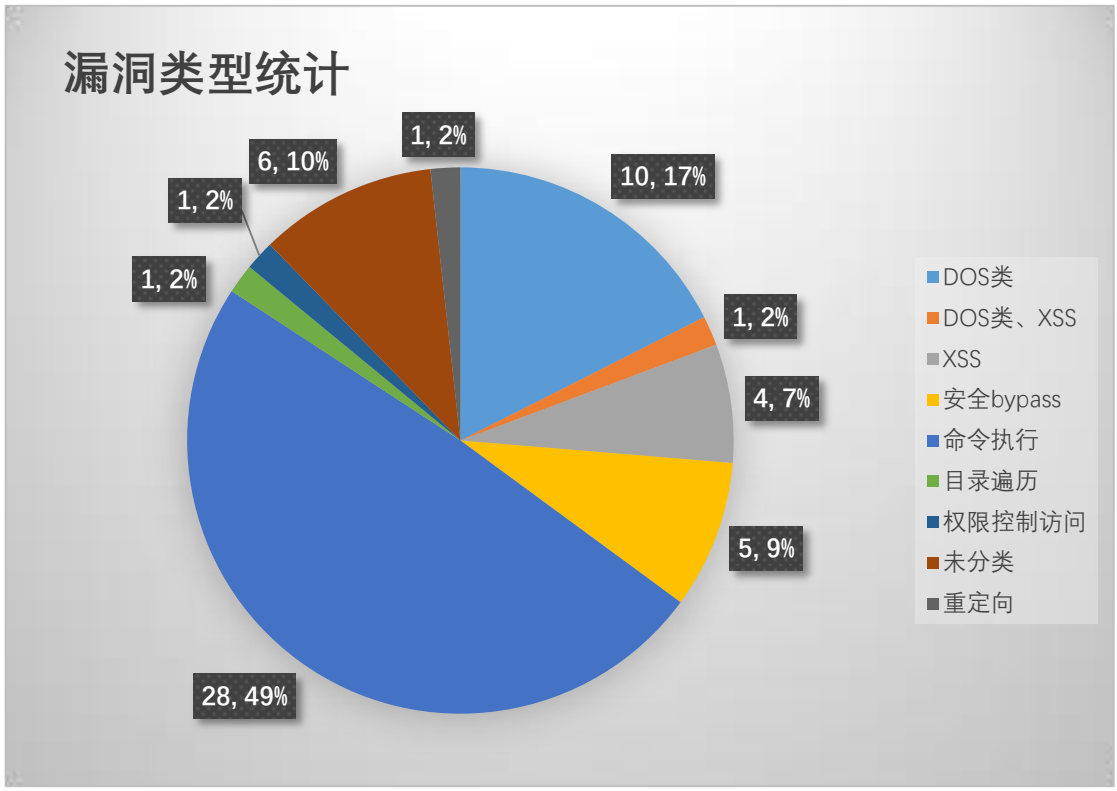


图 4-2 漏洞类型统计

4.3 Struts2 行业分布态势

从近几年天融信所监测的网站中，随机抽取 20000 个做分析。其中存在 Struts2 漏洞的网站行业分布比较广泛，教育行业占比最高（占 25%），其次是政府（占 21%）、金融（占 15%）、企业（占 11%）、能源（占 9%）等行业。

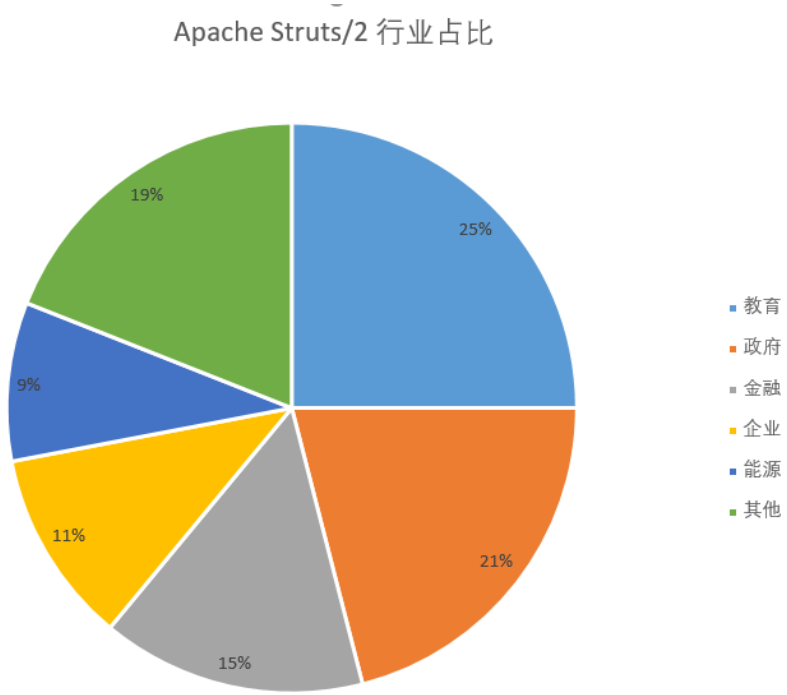


图 4-3 Struts2 漏洞行业分布

4.4 Struts2 全国分布态势

天融信对我国境内使用 Struts2 的 8 万网站进行了抽样统计及分析，其中设备数量最多的省份是北京市大约 18895 台，占全国数量的 21%；其次是广东省 15243 台，占全国数量的 17%；浙江省 11954 台，占全国数量的 14%。

国内分布

图 4-4 Struts2 国内分布图

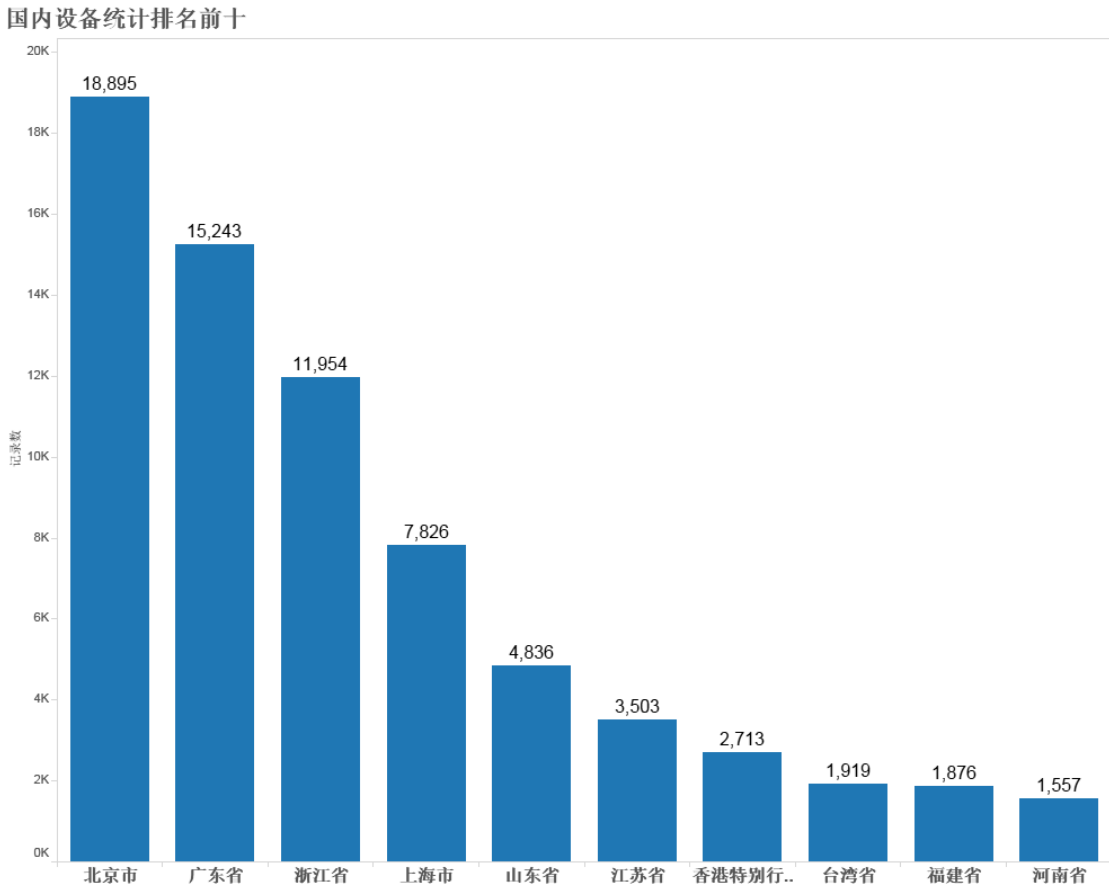


图 4-5 国内设备统计排名前十

北京市、广东省、浙江省排名前三。浙江省和广东省作为经济大省，同时信息化的发展也处于全国前列。其中北京市作为中国的首都，政治、文化的中心，高校云集，政府部门集中，由图 3-3 的行业分布可以看到，政府、教育部门分布较多，与 Struts2 在行业的分布情况一致。

4.5 Struts2 运营商分布态势

我国范围内 Struts2 网站运营商分布情况统计排名前十如图 3-6 所示：

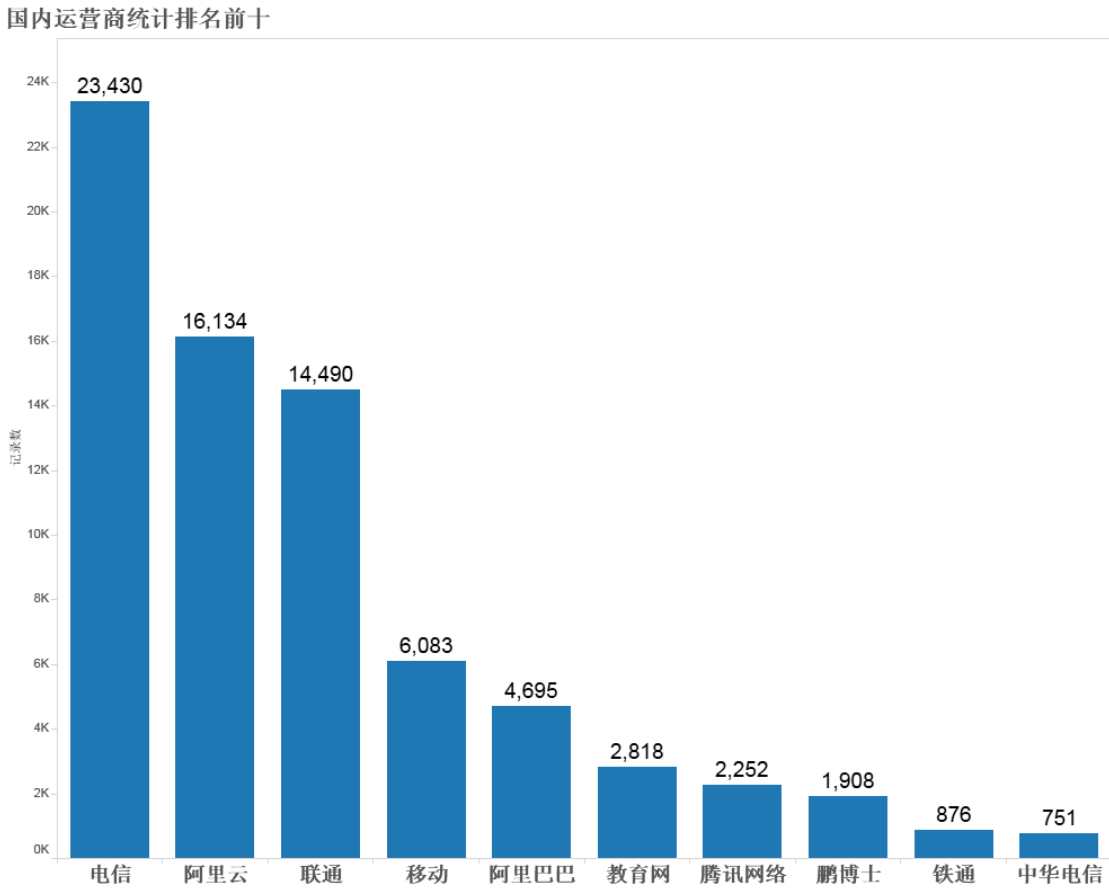


图 4-6 运营商统计排名前十

排名前三为的运营商分别是电信、阿里云、联通，这与我国互联网运营商基础设施占比基本一致。

4.6 端口开放情况统计

我国范围内使用 Struts2 组件的服务器端口开放情况统计如图：

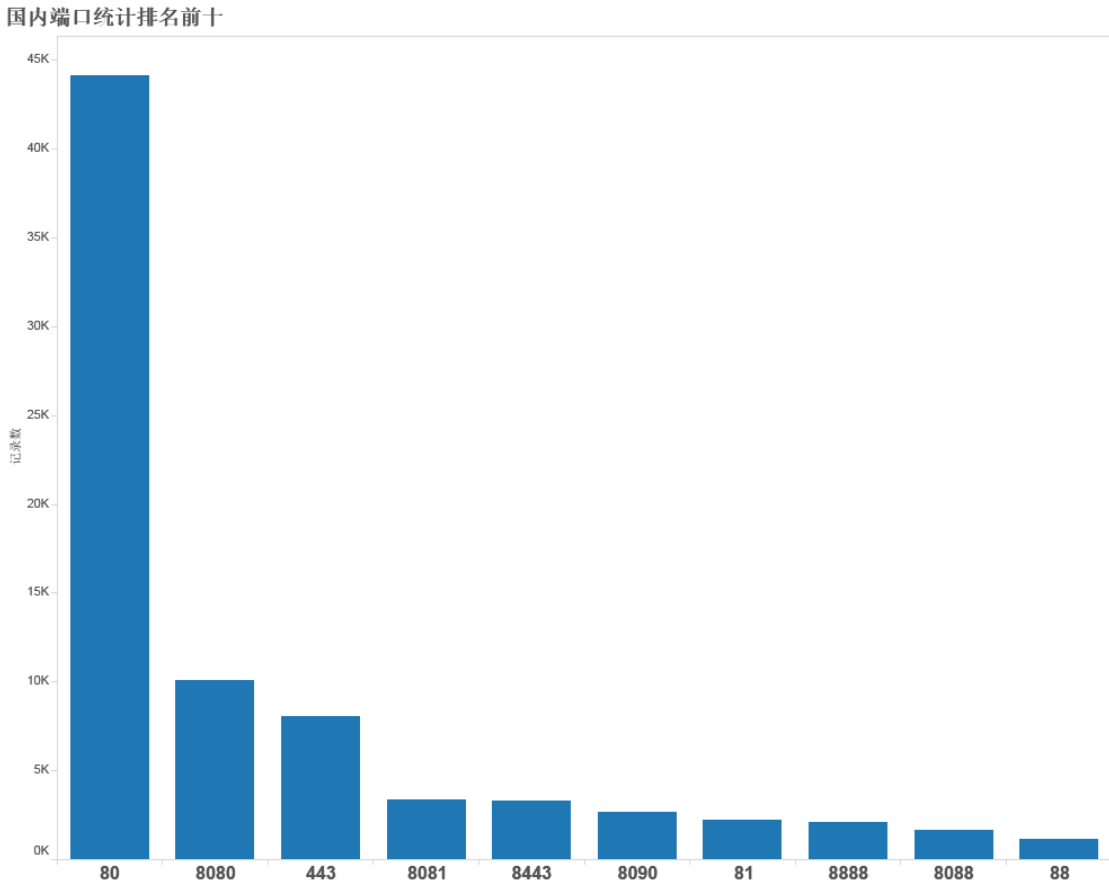


图 4-7 国内设备端口统计排名

从图 3-7 可以看出， Struts2 的服务器开放的主要是 80、443、8080 等常用端口。

4.7 行业应对漏洞积极性

从应对漏洞积极性来说，金融、政府、教育位列前三甲。金融行业应急反应最为迅速，在漏洞爆发后采取行动也是最迅速的，无论是自行升级漏洞软件还是联系厂商升级防护设备都走在其他行业前列，很多金融行业站点在几个小时之内再次扫描时已经将漏洞修补完成。

4.8 Struts2 漏洞真实案例

2014 年 9 月，《法制晚报》报道过一个案例，某公司安全测试工程师王某，在看到 Apache 基金会公布的漏洞之后，利用该漏洞入侵了 某 邮箱的服务器，将该邮箱的 1.6 万多家企业用户的数据全部拿下。

2016 年 12 月，某知名电商被曝出 12G 用户数据泄露，一个 12G 的数据包在地下渠道流通，其中包括用户名、密码、邮箱、QQ 号、电话号码、身份证等多个维度，数据多达数千万条。对此，京东回应称：该数据的泄露是源于 2013 年 Struts 2 的安全漏洞问题。

而不仅仅是邮箱和电商受到了危害，其实国内几乎所有互联网公司及大量银行、政府机构都受到了影响。

五 总结及建议

5.1 总结

5.1.1 漏洞危害情况总结

根据上面的图形数据和 10 年的漏洞跟踪，Struts2 漏洞大致可分为 DOS、XSS、安全 bypass、命令执行、目录遍历、权限控制访问、重定向和其他未有明确分类的种类。

命令执行执行类漏洞所造成的伤害最大也最为直接，例如编号 S2032，当启用动态方法调用时，攻击者可以构造恶意表达式，直接可在服务器端执行任意代码，获取服务器权限。编号 S2048，当你的 Web 应用使用了 Struts 2 Struts 1 插件，则可能导致 Struts2 执行由外部输入的恶意攻击代码，对用户数据造成的是毁灭性的打击。

XSS 类漏洞可劫持用户 cookie，通过在网页中写入并执行脚本执行文件，劫持用户浏览器，将用户当前使用的 sessionID 信息发送至攻击者控制的网站或服务器中。或者是操作网页中的 DOM 树结构和内容，在网页中通过 JS 脚本，生成虚假的页面，欺骗用户执行操作，而用户所有的输入内容都会被发送到攻击者的服务器上。如编号 S2006，XWork 默认未过滤 action 来进行 XSS 攻击。

目录遍历类漏洞，允许攻击者遍历目录结构和相对路径下载“静态”内容文件夹之外的文件，对应编号 S2004。

安全 bypass 类漏洞，攻击者通过滥用已知的会话属性来绕过令牌检查，来执行 CSRF 攻击，对应编号 S2010。

DOS 类漏洞，攻击者可让目标计算机或网络无法提供正常的服务或资源访问，使目标系统服务停止响应甚至崩溃，这些服务资源包括网络带宽，文件系统空间容量，开放的进程或者允许的连接，对应编号 S2-011。

Struts2 漏洞自 2007 年以来就一直频繁爆发，原因是该漏洞通常都是以黑名单的方式进行修复，这就导致了层出不穷的绕过。同时，作为全球最流行的框架之一，Struts2 漏洞一直以来也备受关注，无论是黑客，开发，运维，还是安全人员都非常重视该漏洞。攻与防的较量从未停止，在 Struts2 框架漏洞这个战场上，只有持续深入地研究，才能占有主动权。

5.1.2 数据分析总结

从 2007 年第一个 Struts2 漏洞爆发开始，漏洞数量逐年递增，16 年更是达到了顶峰有 18 个之多，17 年数量有所减少但是也有 11 个之多，横跨 10 年，57 个 Struts2 漏洞中，命令执行类漏洞占比最高约为三分之一，数量达到了 28 个，DOS 类漏洞屈

居第二，数量也有 11 个。2012 年-2017 年，Struts2 高危漏洞呈现出频发状态，其类型主要为代码执行或者命令执行，不管是 2012 的 CVE-2012-0838，黑客可以通过 OGNL 表达式实现代码注入和执行，还是 2017 的 CVE-2017-5638，在解析器执行文件上传的时候造成的代码执行。期间的每一个安全漏洞都可能造成巨大的风险，波及的版本范围从 Struts2 2.0.x 到 2.5.x。

经天融信安全云服务中心所做的抽样数据统计分析来看，北京市，广东省，浙江省使用 Struts2 的网站的情况明显高于其他省份。北京市约占到抽查总量的 21%，广东省约占 17%，浙江省约占到 14%。其中运营商多为电信，阿里云和联通，这与我国互联网运营商基础设施分配比例基本一致。

5.1.3 行业地区影响总结

从行业分布图上来看，教育行业使用量最多，受到的影响也是最多的约占抽查总量的 25%，排名第二的是政府行业，占 21%，其次是金融，企业，能源等行业。可见 Struts2 漏洞的爆发，基本对于各行各业都产生了影响。

5.2 建议

Struts2 的使用群体平时需多关注官方公告，以及各大安全厂商的预警分析通告。如果你的设备已经检测出存在 Struts2 漏洞，根据您的具体情况有以下两种解决方式：

1.官方解决方案

这里给出了 apache 厂商安全公告链接地址：https://cwiki.apache.org/confluence/display/WW/Security+Bulletins，可随时关注相关信息，如已经发布版本更新，尽快升级到不受影响的新版本，建议在升级前做好数据备份。

2.技术解决方案

对于没有网络防护设备的企业，可以使用专业厂商的防护设备进行防护或者使用专业安全厂商的针对性安全服务对已有业务进行漏洞排查和修复。正在使用安全防护设备的企业，目前各大安全厂商都已经推出针对该漏洞的紧急升级包，请及时升级已有防护设备的防护规则和检测规则，来避免漏洞所造成的危害。