某NC系统 ActionInvokeService反序列化任意方法调用漏洞分析

作者： xsshim

在6月份时，某NC系统更新了一个补丁

关于NC系统的命令执行漏洞ActionInvokeService的安全通告

漏洞概述

一、漏洞描述

通过调用ActionInvokeService.class方法接口进而调用到Execute.class可执行任意命令。利用此漏洞黑客可直接破坏软件系统、获取服务器权限、植入木马病毒，或配合内网穿透工具间接获取数据库权限、获取客户敏感数据。

二、影响版本

NC63 / NC633 / NC65

看描述，问题显然是出现在了 ActionInvokeService 这个接口上。

既然知道了类名，那就全局搜索一下咯

发现在 `home\modules\aert\META-INF\query.upm` 这个文件下，出现了这个类名

```
<?xml version="1.0" encoding='UTF-8'?>
<module name="aert">
    <public>
        <component remote="true" singleton="true" tx="NONE">
            <interface>com.ufida.zior.console.IActionInvokeService</interface>
            <implementation>com.ufida.zior.console.ActionInvokeService</implementation>
        </component>
        <component cluster="SP" remote="true" singleton="true" accessProtected="false" tx="N
```

那么我们直接找个地方 import 一下，直接就可以跳到这个class 处

```
package com.ufida.zior.console;

import ...

public class ActionInvokeService implements IActionInvokeService {
    public ActionInvokeService() {
    }

    public Object exec(String actionName, String methodName, Object paramter) throws Exception {
        if (Logger.getModule() == null) {
            Logger.init( name: "iufo");
        }

        AppDebug.debug( msg: DateUtil.getCurTimeWithMillisecond() + " ActionInvoke: " + actionName + "." + methodName + "()");
        return ActionExecutor.exec(actionName, methodName, paramter);
    }
}
```

跟进exec处

```
final class ActionExecutor {
    private static final Map<String, Method> map_method = new ConcurrentHashMap();

    ActionExecutor() {
    }

    static Object exec(String actionName, String methodName, Object paramter) throws Exception {
        if (actionName != null && methodName != null) {
            Object action = Class.forName(actionName).newInstance();
            int paramNum = false;
            int paramNum;
            if (paramter == null) {
                paramNum = 0;
            } else if (paramter.getClass().isArray()) {
                paramNum = ((Object[])((Object[])paramter)).length;
            } else {
                paramNum = 1;
            }
```

actionName通过反射的方法实例化了一个action，还会判断paramter是数组还是是string类型

```
        String key = actionName + ":" + methodName + ":" + paramNum;
        Method m = (Method)map_method.get(key);
        Class actionclz;
        Class[] types;
        Object[] arr;
        if (m == null) {
            actionclz = action.getClass();

            int len$;
            try {
                types = new Class[]{Object.class};
                if (paramter != null && paramter.getClass().isArray()) {
                    arr = (Object[])((Object[])paramter);
                    List<Class<?>> clzList = new ArrayList();
                    Object[] arr$ = arr;
                    len$ = arr.length;

                    for(int i$ = 0; i$ < len$; ++i$) {
                        Object obj = arr$[i$];
                        if (obj == null) {
                            clzList.add(Object.class);
                        } else {
                            clzList.add(obj.getClass());
                        }
                    }

                    types = (Class[])clzList.toArray(new Class[clzList.size()]);
                }

                m = actionclz.getMethod(methodName, types);
            } catch (NoSuchMethodException var15) {...}

            if (m != null) {
                map_method.put(key, m);
            }
        }
```

接下来就根据传入的参数到底是 数组 还是 字符串，嘎嘎一顿反射执行

```
            if (m == null) {
                throw new IllegalArgumentException("Mthod " + methodName + " not exists.");
            } else {
                actionclz = null;
                types = m.getParameterTypes();
                Object result;
                if (types != null && types.length >= 1) {
                    if (types.length == 1) {
                        if (paramter != null && paramter.getClass().isArray()) {
                            arr = (Object[])((Object[])paramter);
                            if (arr.length == 1) {
                                result = m.invoke(action, arr[0]);
                            } else {
                                result = m.invoke(action, paramter);
                            }
                        } else {
                            result = m.invoke(action, paramter);
                        }
                    } else {
                        result = m.invoke(action, (Object[])((Object[])paramter));
                    }
                } else {
                    result = m.invoke(action);
                }

                return result;
            }
        } else {
            throw new IllegalArgumentException();
        }
    }
```

那么假如 我的 actionName 是 `bsh.Interpreter`，methodName是 `eval`，

那这岂不是又是一个任意的代码执行漏洞?

再去看看如何调到这个 ActionExecutor呢

在 com.ufida.zior.console.ActionHandlerServlet 可以找到调用这个 ActionExecutor 的接口

```
public class ActionHandlerServlet extends HttpServlet {
    private static final long serialVersionUID = -2965409880483393507L;

    public ActionHandlerServlet() {
    }

    protected void process(HttpServletRequest request, HttpServletResponse response) {...}

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        this.process(request, response);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        this.process(request, response);
    }
}
```

看看process函数，很明显有反序列化漏洞，但这次并不关注这个反序列化，而是着重看如何调用到这个 ActionExecutor.exec() 方法

```java
protected void process(HttpServletRequest request, HttpServletResponse response) {
    ObjectOutputStream out = null;

    try {
        String msg;
        try {
            ObjectInputStream ois = new ObjectInputStream(new GZIPInputStream(request.getInputStream()));
            msg = (String)ois.readObject();
            String methodName = (String)ois.readObject();
            Object paramter = ois.readObject();
            String currentLanguage = (String)ois.readObject();
            String logModule = (String)ois.readObject();
            ois.close();
            MultiLangUtil.saveLanguage(currentLanguage);
            if (logModule == null) {
                logModule = "iufo";
            }

            Logger.init(logModule);
            out = new ObjectOutputStream(response.getOutputStream());
            Object result = ActionExecutor.exec(msg, methodName, paramter);
            out.writeObject(true);
            out.writeObject(result);
        } catch (Exception var21) {
```

所以，我们只要构造出 msg, methodName, paramter 这几个变量，

通过序列化的方式传入，即可触发反射调用任意高危函数。

需要注意的是，这里还用了GZIPInputStream这个函数，相比与之前爆出来的一些原生的反序列化漏洞来说，

这个自身就可以绕过一些waf流量设备。

POC就不放出来了，熟悉反序列化的朋友想构造出来还是比较简单的。