Advertisement Removal of Android Applications by Reverse Engineering

Jen-Yi Pan
Department of Communications Engineering
National Chung Cheng University
Chia-Yi, Taiwan, R.O.C.
jypan@comm.ccu.edu.tw

Sheng-Hao Ma
Department of Information Engineering
I-Shou University
Kao-Hsiung, Taiwan, R.O.C.
aaaddress1@gmail.com

***Abstract*—Millions of apps cover all aspects of our lives and make smartphones success. Advertisers financially support developers but also collect users' privacy. Several ad blocking apps help users to eliminate advertisement by filtering or redirection for certain targets whose list is maintained by a third party, and some of them require root permission. We develop an advertisement removal program with the technique of reverse engineering, which can effectively patch the advertising code, even obfuscated by other tools. The proposed method takes no overhead at runtime, and requires neither root permission nor list updating. This study also gives a different perspective on tailoring behavior of Android applications.**

Keywords—advertisement blocking; Android apps; reverse engineering

I. INTRODUCTION

Success of smartphones comes from availability of millions of apps that offer various functions covering all aspects of our lives. Most of these apps are free to download. Their developers embed advertisement area in the user interface of apps and get financial support from advertisers. To maximize the potential values, advertisers gain better understandings of user habits and behaviors for accurate ad targeting from private user data [1], ranging from the phone's IMEI number, MAC addresses of nearby access points, the user's location, even the contact list. These "free" apps are not entirely free: users pay the price of their privacy.

Generally speaking, there are three major components in the advertising ecosystem [2]. The first one is the advertisement library, which is packaged alongside the app that users download and install. It transmits advertisement requests to the advertisement network and displays the advertisement content after receiving. The second component is the advertisement network (ad network), which is a brokering company with a real time bidding system that dynamically adjust the prices that advertisers pay in order for users to see their advertisement. The final one is the advertiser or the advertisement agency that is willing to pay for potential customers' impression on their advertisement.

App developers first register their app with the ad network and get a unique identifier for each ad space which is embedded in their app. The space can host videos, images, even web pages containing javascripts. An impression is a successful rendering of an advertisement within the app, and the advertiser pays a fee to the ad network for an impression. When users click on the advertisement, the unique identifier also enables the ad network to create profiles for the users, and the ad network can tailor the type and content of future advertisement to increase the ad effectiveness in turning a viewer impression into a sale or action, which is called conversion.

However, hackers often replace advertisement or embed new advertisement to hijack developers' advertising revenue [3]. Developers might not share the fee from users. Users' private and personal information also may be sold to another party without their awareness [2]. If a user notices privacy violations carried out by the advertisement, she cannot restrict the application's behavior without rewriting the application [4].

Different methods of ad blocking and ad blocking tools available for traditional web browsers and mobile devices were investigated in [5]. Methods applicable to Android applications include ad domain name looping back with the hosts file and passing traffic through a content-filtering proxy server, such as adAway[6] and AdBlock[7]. In addition, a few apps limit ad network connection, such as NoRoot Ad-Remover Lite[10]. We propose a new method based on the technique of reverse engineering to remove advertisement on apps, without incurring extra cost at run time and maintaining filtering list. Reverse engineering is the process of extracting design information from the executable [8]. The technique has been applied on research of Android apps such as reasoning on private analysis [3] and exploring symptoms of app repackaging [9]. This study tries to extract the ad coding pattern inside apps and skip a part of code for advertising with an automatic process.

The structure of this paper is as follows. Section II presents related works on advertisement removal. Section III describes our method and discusses some special cases. Section IV shows the result of our implementation and compares with other methods. We finish by concluding in Section V.

II. RELATED WORK

We take four typical ad blocking apps to exemplify related techniques. More details of ad blockers can be found in [5].

978-1-5090-4588-4/17/$31.00 ©2017 IEEE

A. AdBlock Plus

AdBlock [7] is a well-known ad blocker for typical desktop computers. It is free and open-sourced. Other versions for Android mobiles and web browsers are recently also announced. Users can install the AdBlock plug-in for their browsers such as FireFox, Chrome, and Safari. The plug-in checks all the request of uniform resource locators (URL) and refuses those URLs contained in the list of advertising sites, which is maintained by them.

AdBlock Plus is an ad blocker app on Android platform. If the device is "rooted", Adblock Plus will filter all web traffic out of the phone. For a normal device, Adblock Plus needs to be manually configured as a proxy server, which checks all the requests of URL against the black list for all WiFi traffic.

B. AdAway

AdAway [6] is an open source ad blocker for Android using the hosts file, and needs "root" access. It modifies the hosts file located in "/system/etc/hosts" to loop back the traffic with advertiser's domain to the localhost thus blocking the advertisement.

C. NoRoot Ad-Remover Lite

"NoRoot Ad-Remover Lite" [10] is an ad removing app which as the name implies the app requires no root on the android device. This app turns off the network connectivity when the target application is executed at the initial phase, causing the advertisement content cannot be loaded through the network. When the target application starts the main loop, it turns on the network again. However, this method could disrupt some apps that require networks to communicate with servers at the initial phase, such as multi-player games.

D. AppBrain Ad Detector

"AppBrain Ad Detector" [11] detects all annoyances of apps installed on phones. It identifies which ad networks are embedded in apps, such as Admob, Millennial Media, ChartBoost, TapJoy and so on. It also tells users what apps have permissions to access messages or accounts and could thus invade privacy, or can use services which could cost money. However it does not really block advertisement and users only have two choices: using apps with advertisement, or uninstalling the apps.

III. Code Analysis and Proposed Method

At first, the methodology of reverse engineering is briefed in subsection A. Subsection B exemplifies a code template of embedding an advertisement in Android applications and discusses how to patch. Then, we discuss two special cases: the case of C sharp and Unity engine in subsection C and the case of obfuscator ProGuard in subsection D. Finally, subsection E describes the proposed method.

A. Reverse Engineering on Android Platform

Dalvik [12] is a virtual machine (VM) in Google's Android operating system 4.4 and earlier that executes applications written for Android. Programs for Android are commonly written in Java and compiled to bytecode for the Java virtual machine, which is then translated to Dalvik bytecode and stored in classes.dex (Dalvik EXecutable). The successor of Dalvik is Android Runtime (ART) [13], which also uses the same bytecode and .dex files. The new runtime environment was included for the first time in Android 4.4 as a technology preview, and replaced Dalvik entirely in later versions. The Dalvik executable file is then packaged with other resource files into the file format "Android application package" (APK) for future distribution and installation of mobile apps and middleware. An APK file contains all of that program's code (such as .dex files), resources, assets, certificates, and manifest file.

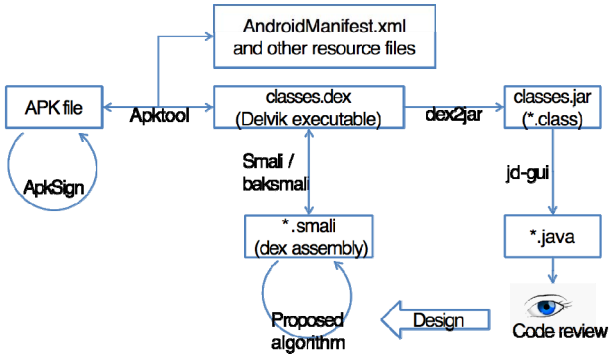


Fig. 1. Reverse engineering process of Android applications.

Fig.1 shows the reverse engineering process. The APK file of an Android application can be unwrapped by Apktool [14], resulting in several files, such as the Delvik executable classes.dex, the manifest file, and other resource files. The tool dex2jar [15] then can convert classes.dex of Dalvik virtual machine into classes.jar of Java virtual machine (JVM). Finally, the tool jd-gui [16] can convert classes.jar to several Java source files of class definition. We can review these source codes and design an automatic program to patch the app. However, the program does not directly patch these Java source files because these decompiled results from JVM bytecode are not complete and compilable Java source codes most of the time. Some information in source codes may be lost in the compiled form, such as names of local variables or parameters, and cannot be recovered through decompiling. Hence, we use the tool baksmali [17] to disassemble the Delvik executable classes.dex to several files in Delvik assembly language. After our program patches these codes, the tool Smali [17] can assemble these codes back to the Delvik executable file again. Finally, we can use Apktool to package the classes.dex into the APK file, and use ApkSign [18] to sign the file, which then can be installed later.

B. Advertisement Code Template and Discussion

The site of Google Developers [19] gives a guideline to embed its ad network AdMob [20] about two types of advertisement: banner ads and full-screen interstitial ads.

For deployment of a banner ad, the developer shall instantiate an object of AdView class, and set its size and unit identifier. Then an ad request is created by creating an object of AdRequest.Builder and initializing its content with method build( ). Finally loadAd( ) of AdView loads the advertisement. The developer also can set the refresh period for the ad, and the ad request object can automatically reload a new ad periodically. Fig.2 shows an example code of banner ads.

Similarly, for deployment of a full-screen ad, the developer shall create an object of InterstitialAd and set its unit identifier. After that, he creates an ad request and loads it, as shown in Fig.3.

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);

        // Create a banner ad.
        // The ad size and ad unit ID must be set before calling loadAd.
        mAdView = new AdView(this);
        mAdView.setAdSize(AdSize.SMART_BANNER);
        mAdView.setAdUnitId("myAdUnitId");

        // Add the AdView to the view hierarchy.
        layout.addView(mAdView);
        // Create an ad request and loading the ad.
        AdRequest.Builder adRequestBuilder = new AdRequest.Builder();
        mAdView.loadAd(adRequestBuilder.build());

        setContentView(layout);
    }
```

Fig. 2. Code example for deploying a banner ad.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Create an interstitial ad.
    // When a natural transition in the app occurs , show the interstitial.
    mInterstitialAd = new InterstitialAd(this);
    mInterstitialAd.setAdUnitId("myAdUnitId");

    // Create an ad request.
    AdRequest.Builder adRequestBuilder = new AdRequest.Builder();
    mInterstitialAd.loadAd(adRequestBuilder.build());
}
```

Fig. 3. Code example for deplying a full-screen interstitial ad.

The reverse engineering allows us to convert app packages into an intermediate form between Delvik bytecode and Java source code. Observing the above process, we have three possible ways to make banner ads disappear in the app.

- Removing codes to create AdView object: This might cause errors for most of apps because they assume the object should exist without checking its availability.
- Setting ad size to zero: This way does hide the ad, but the ad object might refresh periodically and generate network traffic.
- Removing codes of loadAd( ): This is the approach we take in this paper.

We have similarly consideration in full-screen ads. Invalidating the loadAd( ) codes is a common way of dealing these two types of advertisement. In the following, we discuss two special cases which complicate the above observation with reverse engineering on code of applications.

C. *Case of Unity Engine and C Sharp*

Unity [21] is a cross-platform game engine and used to develop video games for desktop computers, mobile devices and websites. However, most applications using Unity are written by C sharp and supported by .NET framework. Common Intermediate Language (CIL) [22] inside the .NET framework complicates our reverse engineering. Fortunately, embedding an advertisement in apps always comes to the class AdView in the file classes.dex, and our program still can clean up the code.

D. *Case of Obfuscator*

ProGuard [23] is a well-known obfuscator and can apply to Android applications. By observing the result of reverse engineering, we know that ProGuard renames every single function inside an app as shown in Fig.4 and cuts the function into many pieces. Then it wraps these pieces in a new class called *internal*, as shown in Fig.5. Reversing the process is a very hard work, and these code pieces are also hard to understand. However, ProGuard does not encrypt strings in the program. Following these plain-text strings, we find a piece of information. To simplify the debugger to track where the bug is, the compiler inserts a name tag of the currently executed function for displaying the error message and function's name while throwing an exception or error, just like the statement b("loadAd") in Fig.5. Hence we can locate the place of these "loadAd" tags, in codes of which a part of loadAd() is contained, and leave the function skeleton in Smali format. Erasing all the parts of loadAd() is equivalent to erasing the whole loadAd() function.

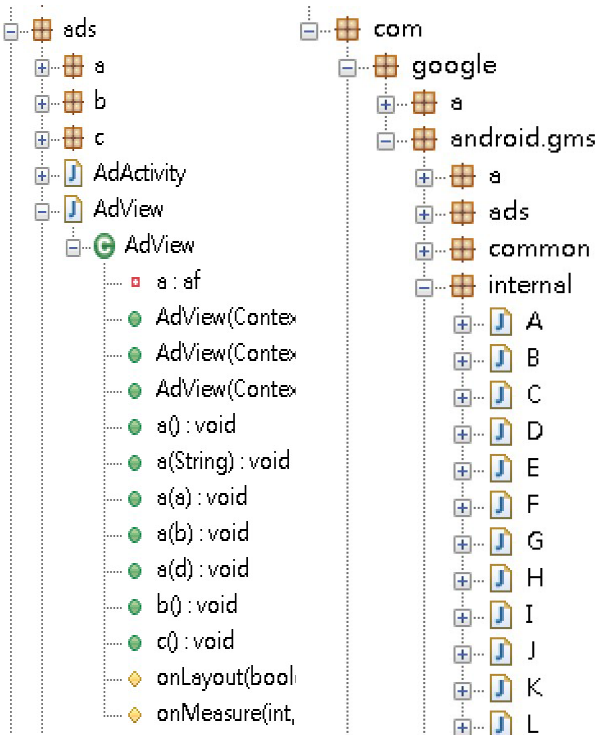


Fig. 4. Class view of obfuscated class AdView.

E. *Proposed Method*

The pseudo code of our proposed method is as follows.

1. Extract classes.dex from the APK file
2. Translate classes.dex to Smali assembly code
3. For all *.smali, find the loadAd() and empty it
4. If protected by ProGuard, find all functions with the string "loadAd" in the internal class and empty those

5. Translate the modified version of Smali code back to classes.dex
6. Repackage classes.dex into the APK file and sign the APK file

We implement an automatic program based on the above, and demonstrate some results in the following section.

```
private void b(String paramString)
{
    if (this.e != null)
      return;
    throw new IllegalStateException(
            "The ad unit ID must be set on InterstitialAd before " +
            paramString +
            " is called." );
}

public final void a(ad paramad) {
    try {
        if (this.e != null)
            return;
        if (this.f == null)
            b("loadAd");
        this.e = M.a( this.b, new ay(), this.f, this.a);

        if (this.d != null)
            this.e.a(new L(this.d));

        if (this.e.a(O.a(this.b, paramad)))
            this.a.a(paramad.i());
    }
    catch (RemoteException localRemoteException)
    {
    }
}
```

Fig. 5. Small pieces of loadAD( ) code inside the class *internal* after obfuscation.

IV. Result and Discussion

A. Implementation and Result

Based on the discussion in subsection III.B, our program shall hollow out the function loadAd( ) in Smali format. Fig.6 gives the Smali code of loadAd( ), and Fig.7 gives the hollowed out example after clearing the function body.

```
.method public loadAd(Lcom/google/ads/AdRequest;)V
    .registers 3
    .param p1, "adRequest"    # Lcom/google/ads/AdRequest;

    .prologue
    .line 605
    iget-object v0, p0, Lcom/google/ads/AdView;->a:Lcom/google/ads/internal/d;
    if-eqz v0, :cond_14

    .line 607
    invoke-virtual {p0}, Lcom/google/ads/AdView;->isRefreshing()Z
    move-result v0

    if-eqz v0, :cond_f
    .line 608
    iget-object v0, p0, Lcom/google/ads/AdView;->a:Lcom/google/ads/internal/d;

    invoke-virtual {v0}, Lcom/google/ads/internal/d;->e()V

    .line 611
    :cond_f
    iget-object v0, p0, Lcom/google/ads/AdView;->a:Lcom/google/ads/internal/d;

    invoke-virtual {v0, p1}, Lcom/google/ads/internal/d;->a(Lcom/google/ads/AdRequest;)V
    .line 613
    :cond_14
    return-void
.end method
```

Fig. 6. Original Smali file of loadAd( ).

After patching the function loadAd(), our program invokes the tool Smali to generate classes.dex and packages it into the APK file. Then our program signs the APK file, which can be installed in the Android platform. Fig.8 shows the before and after screen shots of a sample app being patched by our program.

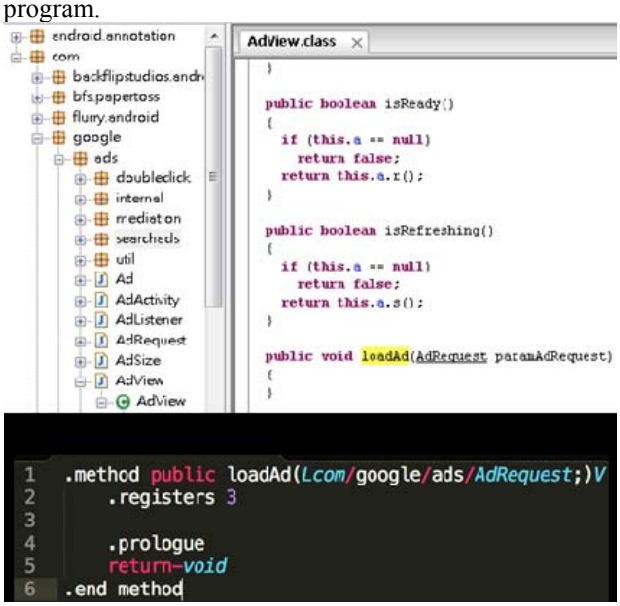
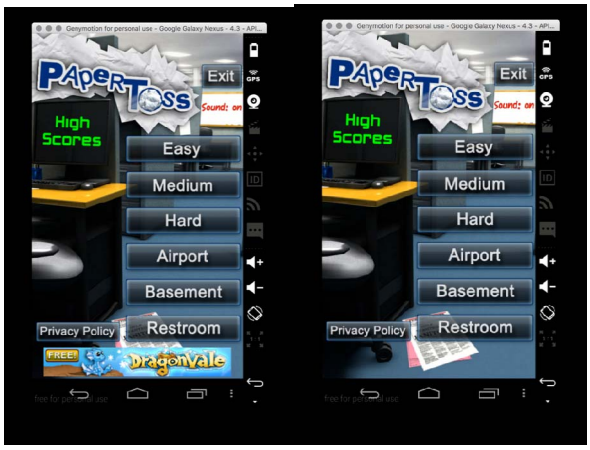


Fig. 7. Patched smali file of function loadAd( ).



(a) Before ad removal (b) After ad removal

Fig. 8. Screen shot of a sample app.

B. Comparison of ad Blocking

We take 15 applications on the Android market and compare few ad blocking methods with our proposed method in Table I, where Android package name is a unique identifier for an Android application in the market. The first five apps are two games, one local app, and two popular utilities for our observation. The middle five apps are from new tool apps in September 2016, and the last five apps are the top five apps from editors' choice [25] in September 2016. The result shows that AdBlock and AdAway with root permission can block well known ads except new apps. AdBlock intercepts all the ad requests and returns empty responses to these requests. AdAway loops back all the ad traffic on the predefined list. Although rooting [24] lets all user-installed applications run privileged commands typically unavailable, rooting is also required for more advanced and potentially dangerous operations including modifying or deleting system files and

Table I. Comparison of methods on advertisement removal.

App name and package name	Version	Google App Store download count			Methods			
			P	U	AdBlock*	AdAway*	No-Root Ad Remover	Proposed
Paper Toss com.bfs.papertoss	1.2.7	50,000,000			V	V	V	V
1010 com.gramgames.tenten	37.0.0	10,000,000		V	V	V	V	V
南台學生助理 (a digital assistant app for students) com.shen.app.iststudentassistant	4.7.0	1,000	V		V	V	V	V
Backgrounds HD Wallpapers App com.ogqcorp.bgh	4.8.7	50,000,000	V		V			
Tiny Flashlight + LED com.devuni.flashlight	5.2.4	100,000,000	V		V	V	V	V
QR Code Reader com.a13softdev.qrcodereader	1.1.5	500,000	V			V		V
Caynax A6W com.caynax.a6w	9.1.0	10,000,000	V			V		V
Pokemon IV Rater com.vincentfungace.pokemonivraterfree	1.1.1	100,000				V	V	V
Speed Checker uk.co.broadbandspeedchecker	2.0.77	1,000,000	V			V	V	V
Unit Converter com.androidapps.unitconverter	2.0.81	1,000,000			V	V		V
(Top#1) Skyscanner Flights, Hotel, Car net.skyscanner.android.main	2.0.21	10,000,000	V					
(Top#2) Unblock Me FREE com.kiragames.unblockmefree	1.5.5.8	50,000,000	V			V		V
(Top#3) Piano+ com.rubycell.pianisthd	20160921	10,000,000	V					V
(Top#4) Erudite Dictionary com.erudite.ecdict	7.17.1	500,000			V	V	V	V
(Top#5) Cookbook Recipes com.riatech.cookbook	11.0.1	1,000,000	V			V		V

P = Obfuscated by ProGuard U = Developed with Unity engine * = root permission required

low-level access to the hardware itself. Thus we suggest that methods with rooting should be prevented. In addition, rooting requires exploiting a known vulnerability in the kernel, which is always patched soon by the Android team.

AdAway and the proposed method does not works on the app “Wallpapers” because the app implements the functionality of loadAd( ) by itself and connects to the ad network until successful communication. During execution, “Wallpapers” periodically loads new advertisements. The “No-Root Ad Remover” also does not work when an application has already been started and always loads new advertisements while execution. Only AdBlock works for “Wallpaper” with generating empty responses.

For the case “Top#1”, it connects to a new ad host which is not on the predefined list, resulting in the uselessness of AdBlock and AdAway. It loads ads while users finish their search, which cannot be blocked by initial blocking of AdRemover. It also checks ads after loading, and hence the proposed method fails. The case “Top#3” is similar to the above, but it only calls loadAd() without further checking. Only the proposed method works on this case.

The proposed method neither requires root permission, nor blocks normal network operation. In addition, we don’t need to install any new apps on the target mobile, and the proposed method takes no extra cost at run time and on maintaining filtering lists. We just explore a technical possibility without considering infringement of original developer's copyright,

and do not encourage any illegal actions. However, the proposed method cannot work on customized code of loading advertisement and also failed in the some cases.

V. Conclusion

This research gives a new perspective of reverse engineering to block advertisement without root permission and runtime overhead. The implementation verifies the feasibility of our proposed method, even under the cases of obfuscation and intermediate codes in apps.

Meanwhile, nowadays development of applications is usually separated into several parts. If a critical part, such as credit card payment or token verification, is integrated in as the way of advertisement library and implemented following a code template, developers should consider that reverse engineering may easily skip this part of code, and should take more care in programming security.

Acknowledgment

This work was supported in part by Ministry of Science and Technology, Taiwan, Republic of China, under grant MOST 105-2221-E-194-011.

References

[1] William Enck, Peter Gilbert, Byung-gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth, “TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones,” USENIX Symposium on Operating Systems Design and Implementation (OSDI), October 2010.

[2] Jeremy Tate, Security Weaknesses of the Android Advertising Ecosystem, LAP LAMBERT Academic Publishing, February 2016.

[3] H. Han, R. Li, J. Hu and M. Qiu, "Context Awareness through Reasoning on Private Analysis for Android Application," 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud), New York, 2015, pp. 148-156.

[4] H. Kuzuno and K. Magata, "Detecting Advertisement Module Network Behavior with Graph Modeling," 2014 Ninth Asia Joint Conference on Information Security (ASIA JCIS), Wuhan, 2014, pp. 1-10.

[5] Doruk Uzunoglu, “Understanding Ad Blockers,” Bachelor Project, Worcester Polytechnic Institute, March 2016. [online] https://www.wpi.edu/Pubs/E-project/Available/E-project-032216-001707/unrestricted/dcuzunoglu_understanding_ad_blockers.pdf

[6] “AdAway,” [online] https://adaway.org/

[7] “Adblock Plus,” [online] https://adblockplus.org/

[8] “Reverse engineering,” [online] https://en.wikipedia.org/wiki/Reverse_engineering

[9] Hugo Gonzalez, Andi A. Kadir, Natalia Stakhanova, Abdullah J. Alzahrani, and Ali A. Ghorbani, “Exploring reverse engineering symptoms in Android apps,” the Eighth European Workshop on System Security (EuroSec '15), ACM, New York, 2015.

[10] “NoRoot Ad-Remover Lite,” [online] http://android-apk.org/apk/com.atejapps.litenorootadremover/1.2/darkpsytrancer/noroot+ad-remover+lite/

[11] “AppBrain Ad Detector,” [online] https://play.google.com/store/apps/details?id=com.appspot.swisscodemonkeys.detector

[12] Wikipedia, “Dalvik (software),” [online] https://en.wikipedia.org/wiki/Dalvik_(software)

[13] Wikipedia, “Android Runtime,” [online] https://en.wikipedia.org/wiki/Android_Runtime

[14] “Apktool,” [online] https://ibotpeaches.github.io/Apktool/

[15] “dex2jar,” [online] https://sourceforge.net/projects/dex2jar/files/

[16] “Java Decompiler,” [online] http://jd.benow.ca/

[17] “Smali/baksmali,” [online] https://github.com/JesusFreke/smali

[18] “Apk Sign,” [online] https://github.com/appium/sign

[19] “Google Developers,” [online] https://developers.google.com/

[20] “AdMob,” [online] https://www.google.com/admob/

[21] Wikipedia, “Unity (game engine),” [online] https://en.wikipedia.org/wiki/Unity_(game_engine)

[22] Wikipedia, “Common Intermediate Language,” [online] https://en.wikipedia.org/wiki/Common_Intermediate_Language

[23] Eric Lafortune, “ProGuard,” [online] http://proguard.sourceforge.net/

[24] Wikipedia, “Rooting (Android OS),” [online] https://en.wikipedia.org/wiki/Rooting_(Android_OS)

[25] “Editors' Choice - Android Apps on Google Play” [online] https://play.google.com/store/apps/collection/editors_choice