Computer Security Symposium 2017
23 - 25 October 2017

Androidアプリケーションの ユーザによる外部ライブラリアップデート手法の提案

小川 弘貴[1]　竹内 俊輝[1]　毛利 公一[2]　齋藤 彰一[1]

概要：Android アプリケーションの開発には，外部ライブラリを利用することが多い．ただし，古いバージョンの外部ライブラリには，脆弱性が存在する可能性がある．そのため，開発者は，外部ライブラリを最新のバージョンに更新すべきである．しかし，Android アプリケーションの中には，外部ライブラリのバージョンを更新していないものがある．そこで本論文では，Android 端末上のアプリケーションに含まれる外部ライブラリを外部サーバと連携してアップデートする手法を提案する．この手法を用いることで，ユーザが，Android アプリケーションに含まれる外部ライブラリをアップデートできるようになる．また，実際のアプリケーションに提案手法を適用し，アプリケーションへの影響を調査する．

キーワード：Android，ライブラリアップデート，外部ライブラリ

1. はじめに

Android アプリケーションの開発において，外部ライブラリは欠かせないものとなっている．開発者は，外部ライブラリを使用することで，開発の効率化や開発速度の向上を図ることができる．主な外部ライブラリの用途として，広告の表示，ソーシャルネットワークサービスなどの認証機能の実装，アプリケーションの使いやすさや利便性を向上させるといったものが挙げられる．このように，Android アプリケーションの開発において，外部ライブラリを使用することには，多くの利点がある．

しかし，外部ライブラリは，Android アプリケーションの開発を助ける反面，セキュリティ上の脅威となる可能性がある．外部ライブラリには，バージョンによって，セキュリティ上の脅威となる脆弱性が存在する場合がある．例えば，Facebook の外部ライブラリでは，バージョンが 3.16 より前のものには，アカウントを乗っ取られる脆弱性が存在する [1]．また，その他にも，Dropbox の外部ライブラリ [2] や Apache CC の外部ライブラリ [3] など，古いバージョンに脆弱性が存在する外部ライブラリは枚挙にいとまがない．そのため，古いバージョンの外部ライブラリを使用することは危険である．よって，開発者は自らが公開したアプリケーションが使用する外部ライブラリを，常に最新に保つべきである．しかし，Android アプリケーションの中には，開発者がアップデートを忘れていたり，開発が終了しているといった理由で，外部ライブラリのバージョンがアップデートされていないものがある [4]．一方，ユーザにはアプリケーションを独自にアップデートする方法が提供されていない．このため，ユーザが外部ライブラリをアップデートしたいと考えた場合でも，アップデートする方法がない．そこで本提案では，ユーザが Android アプリケーションに含まれる外部ライブラリをアップデートできる手法を提案する．

本提案手法は，外部サーバと連携して，Android アプリケーションに含まれる外部ライブラリを特定し，古いバージョンの外部ライブラリを使用していた場合にアップデートを行う．本提案手法により，ユーザは自身で Android アプリケーションに含まれる外部ライブラリをアップデートでき，古いバージョンの外部ライブラリの脆弱性を無くすことができる．

以後，2 章で関連研究について述べ，3 章で提案とその詳細を述べ，4 章で実装について述べる．続いて 5 章で評価を行う．6 章で今後の課題について述べ，7 章でまとめる．

2. 関連研究

本章では，関連研究について述べる．まず，Android アプリケーションに含まれる外部ライブラリを特定する手法について述べる．その後，Android アプリケーションをアップデートする手法について述べ，アプリケーション

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 立命館大学
Ritsumeikan University

アップデート手法の問題について述べる．

2.1 Android アプリケーションに含まれる外部ライブラリ特定手法

Android アプリケーションに含まれる外部ライブラリを特定する手法は，主に，ホワイトリストによる特定手法 [5], [6], [7]，機械学習による特定手法 [8], [9], [10]，クラスタリングによる特定手法 [11], [12] の 3 種類に分けられる．

ホワイトリストによる特定手法では，外部ライブラリのパッケージ名により，Android アプリケーションに含まれる外部ライブラリを特定する．この手法は，高速であり，簡単に適用できるという特徴がある．しかし，名前難読化されている場合は，特定できないという問題がある．

機械学習による特定手法では，Android アプリケーションに含まれる広告用の外部ライブラリを高い精度で特定できる．しかし，この手法は，広告用の外部ライブラリを特定することを目的としている．

クラスタリングによる特定手法では，事前に外部ライブラリをクラスタリングすることで，API の呼び出し回数などの情報を収集する．得た特徴情報に基づいて外部ライブラリを特定する．この手法も，機械学習による手法と同様に，高い精度で外部ライブラリを特定できるという特徴がある．また，どのような用途の外部ライブラリでも特定できる．しかし，特定したい外部ライブラリを使用している Android アプリケーションが少ない場合，特徴情報が不十分となるため特定が困難となる．

上記の 3 種類以外の手法として，事前に外部ライブラリのクラス階層情報を抽出し，データベースに保存しておき，それを基に特定を行う LibScout[4] がある．また，クラスタリングによる特定手法を拡張し，外部ライブラリの Android API の呼び出し頻度を抽出し，それを基にクラスタリングを行う LibRadar[13] がある．

2.2 Android アプリケーションのアップデート手法

Android アプリケーションのアップデート手法として，Hotfix[14] がある．Hotfix とは，開発者が，迅速にバグを修正した Android アプリケーションを提供できるようにするための手法である．開発者は，バグを修正するスクリプトを用意する．そして，ユーザはそのスクリプトをダウンロードし，実行することで，Android アプリケーションのバグを修正する．

2.3 アプリケーションアップデート手法の問題

既存のアプリケーションアップデート手法では，ユーザが独自にアプリケーションのアップデートを行うことができない．そのため，外部ライブラリや API の脆弱性が公開されても，ユーザはアップデートをする方法がない．また開発者は，常に使用している外部ライブラリや API の脆弱性情報に気を配り，迅速にアップデートを行う必要があり，負担が大きい．よって，ユーザが独自にアプリケーションをアップデートできる手法が必要である．

3. 提案

本章では，まずアップデート手法の検討について述べる．その後，Android 端末上のアプリケーションに含まれる外部ライブラリを外部サーバと連携してアップデートする手法の概要とその詳細について述べる．

3.1 アップデート手法の検討

Android アプリケーションに含まれる外部ライブラリをアップデートするには，当該アプリケーションの配布に用いられるファイルである apk ファイルからその外部ライブラリ及びバージョンを特定する機能が必要である．まず，外部ライブラリのアップデートは，Android アプリケーションに含まれる外部ライブラリを置き換えることで行う．よって，apk ファイルをデコンパイルする機能が必要である．そして，外部ライブラリアップデート後，apk ファイルをリコンパイルする機能が必要である．ただし，apk ファイルをリコンパイルすると署名されていない状態になるため，apk ファイルに署名を行う機能が必要になる．加えて，外部ライブラリを最新バージョンに置き換えるため，最新バージョンの外部ライブラリを所持しておく必要がある．

上記の必要な機能を実現するためには，いくつか必要な要件がある．apk ファイルから外部ライブラリ及びそのバージョンを特定する機能を実現するためには，外部ライブラリの特徴などを集めたデータベースを保持する必要がある．そのデータベースを基に外部ライブラリを特定する．また，apk ファイルのデコンパイル，リコンパイル，署名を行うためのツールを用意する必要がある．そして，外部ライブラリアップデートのため，最新バージョンの外部ライブラリを集めて，保持しておく必要がある．

以上の必要な要件を満たすための性能及び実装手法を検討した結果，アップデート手法を Android 端末上のみで行うことは実装上の困難が伴うため，提案手法のプロトタイプ実装として外部サーバで行う方式とする．なお，アップデート手法をすべて Android 端末上で実現することは今後の検討課題である．

3.2 提案の概要

Android 端末上の Android アプリケーションに含まれる外部ライブラリをアップデートする手法を提案する．本提案手法を利用することで，ユーザが自身で Android アプリケーションに含まれる外部ライブラリをアップデートすることが可能になる．本提案手法は，外部サーバと連携して，

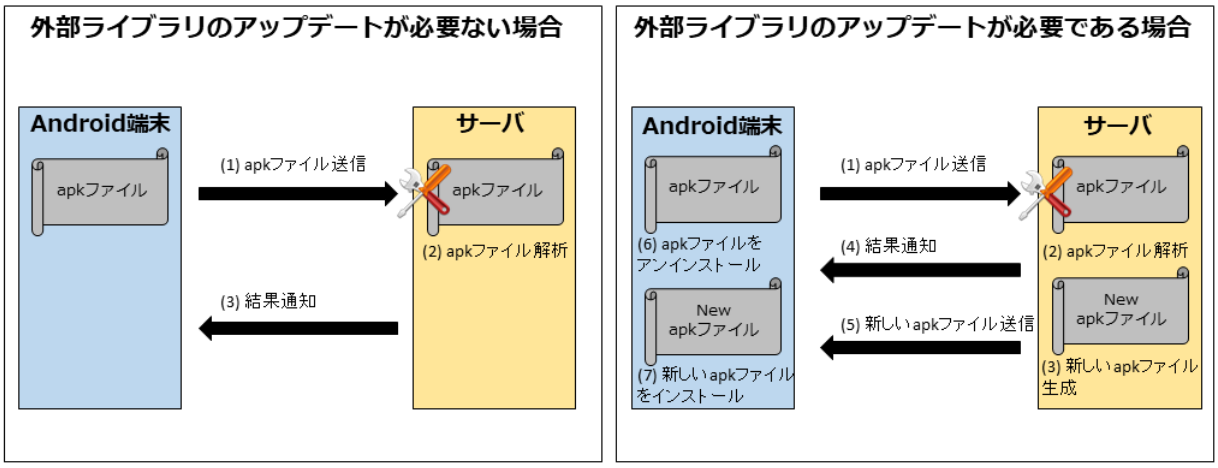

図 1 提案手法の処理の流れ

外部ライブラリをアップデートする．本提案手法の動作概要図を**図 1** に示す．

まず外部ライブラリのアップデートが必要なかった場合の動作を以下に示す．

(1) 外部ライブラリをアップデートしたい Android アプリケーションの apk ファイルを Android 端末から外部のサーバへ送信
(2) サーバで，apk ファイルを解析し，使用されている外部ライブラリとそのバージョンを特定
(3) Android 端末側へアップデートが必要ないことを通知

次に，外部ライブラリのアップデートが必要であった場合の動作を以下に示す．

(1) 外部ライブラリをアップデートしたい Android アプリケーションの apk ファイルを Android 端末から外部のサーバへ送信
(2) サーバで，apk ファイルを解析し，使用されている外部ライブラリとそのバージョンを特定
(3) ライブラリアップデートを行い，新たな apk ファイルを生成
(4) Android 端末へアップデートを行ったことを通知
(5) Android 端末へ新しく生成した apk ファイルを送信
(6) Android 端末は，アップデート前のアプリをアンインストール
(7) サーバから受信した apk ファイルをインストール

3.3 外部ライブラリ特定手法

本提案手法では，Android アプリケーションに含まれる外部ライブラリとそのバージョンを特定するため，2.1 節で述べた LibScout を用いた．LibScout を用いた理由は以下の 3 つである．

- Github でソースコードが公開されている
- データベースへの新たなる外部ライブラリの情報の追加を行う機能がある
- ライブラリ特定の結果において，バージョンまで特定できたものとライブラリ名のみ特定できたものを判別可能である

以上の理由から，本提案手法では，LibScout を用い，外部ライブラリの特定を行う．

3.4 ライブラリアップデート

本提案手法における，Android アプリケーションに含まれる外部ライブラリのアップデート方法は，smali ファイルの置き換えである．smali ファイルとは，Dalvik VM のバイトコードをディスアセンブルしたものであり，apk ファイルに含まれる smali ファイルを置き換えることで，コードを改変できる．また smali ファイルはパッケージごとに別々のディレクトリに保存されている．よって，本提案手法では，外部ライブラリのパッケージ名を基に置き換える smali ファイルの場所を特定する．

本提案手法でのライブラリアップデートの手順として，まず事前に最新バージョンの外部ライブラリを smali ファイルに変換する．外部ライブラリをアップデートする必要がある場合，apk ファイルをデコンパイルし，古いバージョンの外部ライブラリの smali ファイルを，事前に用意した最新のバージョンの smali ファイルに置き換える．最後に apk ファイルをリコンパイルし，署名を行う．以上の処理により，ライブラリアップデートを行う．

4. 実装

本章では，提案手法の機能検証を行うために Android エミュレータ上に構築したプロトタイプシステムについて述べる．提案手法は，Android 端末とサーバとの連携システムであるため，まず Android 側の実装を述べ，その後サーバ側の実装について述べる．

4.1 Android 端末側の実装

今回は，Android 端末側で必要な動作を行う更新処理アプリケーションを作成することで，Android 端末側の実装を行った．更新処理アプリケーションの機能を以下に示す．なお，更新処理アプリケーションはインターネットに接続するための権限が必要である．

- Android 端末にインストールされている apk ファイルの送信
- ライブラリアップデートした apk ファイルの受信
- ライブラリアップデート前の apk ファイルのアンインストール
- ライブラリアップデート後の apk ファイルのインストール

ユーザは更新処理アプリケーションを Android 端末へインストールして，使用することで，Android アプリケーションに含まれる外部ライブラリをアップデートすることができる．アップデート対象の Android アプリケーションは，ユーザがそのパッケージ名を入力することで選択する．更新処理アプリケーションは，Package Manager クラスを使用し，入力されたパッケージ名から選択された Android アプリケーションの apk ファイルのパスを取得する．その後，更新処理アプリケーションが，apk ファイルを外部サーバへ送信及びアップデートされた apk ファイルの受信を行う．またアップデートされた apk ファイルの受信については，外部サーバから通知された外部ライブラリ特定結果により判断する．

外部ライブラリ特定結果より，外部ライブラリのアップデートの必要がないことがわかった場合，更新処理アプリケーションは外部サーバとのコネクションを切断する．そして，再びユーザからのアップデートしたい Android アプリケーションのパッケージ名の入力を待つ．

外部ライブラリのアップデートが必要であることがわかった場合，更新処理アプリケーションは，外部サーバからアップデートされた apk ファイルを受信する．受信が終わると，外部サーバとのコネクションを切断する．その後，アップデート前の apk ファイルを Android 端末からアンインストールする．これは，Android 端末には，同じパッケージ名の apk ファイルを二つ以上インストールすることができないためである．提案手法のライブラリアップデートでは apk ファイルのパッケージ名は変わらない．よって，アップデートされた apk ファイルをインストールするために，アップデート前の apk ファイルをアンインストールする必要がある．アンインストールが終わると，アップデートされた apk ファイルを Android 端末へインストールする．apk ファイルのアンインストール及びインストールは，更新処理アプリケーションがインテントを発行することで行う．アンインストールを行う場合は，ACTION_DELETE を指定し，インストールを行う場合は，ACTION_VIEW を指定する．インストールが終わると，再びユーザからのアップデートしたい Android アプリケーションのパッケージ名の入力を待つ．

4.2 サーバ側の実装

サーバ側で実装する必要がある動作を以下に示し，それぞれ説明する．また今回は，サーバの OS には Linux(Ubuntu 14.04 LTS) を使用する．

- apk ファイルの送受信
- 外部ライブラリ特定
- ライブラリアップデート

サーバは，更新処理アプリケーションからのコネクション接続要求を待つ．更新処理アプリケーションとのコネクションが確立できると，アップデート対象の Android アプリケーションの apk ファイルを受信する．機能拡張した LibScout により，その apk ファイルを解析し，古いバージョンの外部ライブラリを最新バージョンにアップデートする．

LibScout は，事前に収集しておいた外部ライブラリのクラス階層情報と外部ライブラリ特定対象の Android アプリケーションのクラス階層情報を比較することで，外部ライブラリを特定する．また LibScout では，バージョンまで特定できたものとライブラリ名のみ特定できたものを判別できる．よって，バージョンまで特定できたものをアップデート対象とする．今回の実装では，Github で公開されていたデータベースに，いくつかの外部ライブラリのクラス階層情報を追加したデータベースを使用した．新たな外部ライブラリのクラス階層情報は，LibScout の機能を使用することで，データベースへ追加できる．

今回の実装で LibScout に追加した機能は，apk ファイルのデコンパイル，リコンパイル，署名及び smali ファイルの置き換えである．機能拡張した LibScout は，古いバージョンの外部ライブラリを特定できた場合，apk ファイルをデコンパイルする．そして，デコンパイルした apk ファイル内の外部ライブラリの smali ファイルを最新バージョンの外部ライブラリの smali ファイルに置き換える．その後，apk ファイルのリコンパイルを行う．ここで，リコンパイルした apk ファイルは署名されていない状態になる．Android 端末へインストールできるようにするため，署名を行う必要がある．署名ファイルは，事前にサーバで本提案手法用に作成したものを使用した．署名の適切な提供方法については，今後の課題である．また，apk ファイルのデコンパイル及びリコンパイルには apktool を，apk ファイルの署名には，jarsigner をそれぞれ使用した．最後の拡張機能として，apk ファイルのアップデートの処理の有無を LibScout 実行者に通知する．

最新バージョンの外部ライブラリの smali ファイルについては，事前に最新バージョンの外部ライブラリの jar ファ

表 1 Android 端末側の評価環境

エミュレータ	Android Emulator
エミュレータ OS	AndroidOS 7.1.1

表 2 サーバ側の評価環境

提案手法のサーバ動作 OS	Ubuntu 14.04LTS
提案手法開発言語	Java 1.8.0

イルを入手し，それを smali ファイルに変換する．ただし，jar ファイルから smali ファイルへは直接変換できないので，一度 dex ファイルへ変換してから，smali ファイルへ変換する．

サーバと更新処理アプリケーションの通信は，機能拡張した LibScout が，外部ライブラリをアップデートしなかった場合，更新処理アプリケーションにそれを伝えて処理を終了する．外部ライブラリをアップデートした場合，その旨を更新処理アプリケーションに伝え，アップデートした apk ファイルを送信する．送信が終わるとサーバは終了する．

5. 評価

本章では提案手法の評価について述べる．提案手法を使用した，Android アプリケーションに含まれる外部ライブラリ特定についての調査及び提案手法を適用した Android アプリケーションへの影響を調査した結果を示し，考察を行う．

5.1 評価方法

提案手法を用い，Android アプリケーションに含まれる外部ライブラリを特定できるかの調査とライブラリをアップデートした Android アプリケーションへの影響について評価を行う．また，本評価では，Android アプリケーションが動作をするか否かの判定を Android エミュレータを用い確認する．Android 端末側とサーバ側で評価に使用した機器をそれぞれ**表 1**，**表 2** に示す．また，評価には Fossdroid[15] で公開されている**表 3** に示す 5 個の OSS の Android アプリケーションを用いた．Android アプリケーションに含まれている外部ライブラリについては，build.gradle の dependencies タグの中を確認することで特定した．評価に使用した Android アプリケーションを選んだ理由は，最新バージョンの外部ライブラリを使用していなかったことである．

5.2 外部ライブラリ特定に関する調査

提案手法により，Android アプリケーションに含まれる外部ライブラリを特定できるか調査した結果を**表 4** に示す．WiGLE WiFi Wardriving FOSS 以外のアプリでは，外部ライブラリを特定できた．そのため，WiGLE WiFi Wardriving FOSS を手動でデコンパイルし，さらなる調査を行った結果，WiGLE WiFi Wardriving FOSS には slf4j のコードが存在していなかった．これは，slf4j がロギング用のライブラリであるため，リリース版では使用されておらず，最適化などで削除されていると推測される．

5.3 Android アプリケーションへの影響調査

Android アプリケーションに対し，本提案手法を適用したことによる影響を調査した結果を**表 5** に示す．まず，いずれの Android アプリケーションも，リコンパイルと起動が成功することが確認できた．しかし，評価に用いた 4 つの Android アプリケーションのうち，2 つが操作中に強制終了した．この原因を確認するため，強制終了が発生する操作についてさらなる調査を行った．その結果，Veterondo と Capitole du Libre はどちらも，外部ライブラリが使用されると思われる操作を行うと強制終了することが確認できた．これは，Android アプリケーションに含まれる外部ライブラリのバージョンが，最新バージョンに比べ，古すぎることが原因であると考えられる．

そこで，バージョンを最新ではなく，現在のバージョンに最も近く新しいバージョンにアップデートした場合の調査を行った．まず，Veterondo に含まれる外部ライブラリである retrofit を 1.9.0 から 2.0.0 にアップデートし，最新バージョンにアップデートした場合に強制終了が発生する操作を行った．その結果，最新バージョンにアップデートした場合と同様に操作中に強制終了した．その時のログを調査してみると NoClassDefFoundError が発生していた．これは，retrofit が 2 系になると，パッケージ名の retrofit が retrofit2 に変わっていることが原因であると考えられる．続いて，Capitole du Libre に含まれる外部ライブラリである okhttp を 3.4.2 から 3.5.0 にアップデートし，最新バージョンにアップデートした場合に強制終了が発生する操作を行った．その結果，Veterondo と同様に操作中に強制終了した．その時のログを調査してみると NoSuchMethodError が発生していた．これは，バージョン 3.5.0 ではバージョン 3.4.2 の Capitole du Libre が使用していたメソッドがなくなっていると推測される．

以上の調査結果より，ライブラリのバージョンが古すぎる場合，最新バージョンとの違いが大きく，変数やメソッドが削除されている可能性がある．そのため，削除されているメソッドなどを呼び出すと，強制終了することがわかった．また，最新バージョンではなくとも，バージョン間での差異が大きい場合も同様に，強制終了が発生することがわかった．以上より，本提案手法では，Android アプリケーションに含まれる外部ライブラリのバージョンが，最新のバージョンに比べ，古すぎたり，差異が大きいとライブラリアップデートに失敗することがわかる．しかし，本提案手法は，古い外部ライブラリの脆弱性を無くすことが目的である．よって，最新のバージョンではなく，脆弱

表 3 アプリケーションの種類

アプリ名	分類名	アプリに含まれている外部ライブラリ	外部ライブラリのバージョン	外部ライブラリの最新バージョン
PDF Creator	PDF 作成アプリ	itextpdf[16]	5.5.10	5.5.11
Kinolog	動画取得アプリ	parceler[17]	1.1.8	1.1.9
Veterondo	天気予報アプリ	retrofit[18]	1.9.0	2.3.0
Capitole du Libre	情報収集アプリ	okhttp[19]	3.4.2	3.8.4
WiGLE WiFi Wardriving FOSS	WiFi アクセスポイント表示アプリ	slf4j[20]	1.7.19	1.7.25

表 4 ライブラリ特定結果

アプリ名	ライブラリ特定
PDF Creator	成功
Kinolog	成功
Veterondo	成功
Capitole du Libre	成功
WiGLE WiFi Wardriving FOSS	失敗

表 5 提案手法適用による影響の調査結果

アプリ名	リコンパイル	起動	動作
PDF Creator	成功	成功	正常に動作
Kinolog	成功	成功	正常に動作
Veterondo	成功	成功	操作中に強制終了
Capitole du Libre	成功	成功	操作中に強制終了

性を修正したバージョンにアップデートするといった対策をとることで，このような問題を解決できるため，問題ないと考える．また，バージョンによる違いがバグフィックスなどの修正のみであり，バージョン間の差異が小さい場合，ライブラリアップデートが成功する可能性は高くなると考えられる．PDF Creater と Kinolog の 2 つの評価結果がこれに該当すると考える．

6. 難読化対応

難読化された外部ライブラリへの対応が，今後の課題として挙げられる．本提案手法では，外部ライブラリが難読化されている場合を考慮していない．そのため，外部ライブラリが難読化されているとライブラリアップデートを行うことができない．また外部ライブラリのバージョンまで特定することも困難になる．よって，難読化された外部ライブラリの特定精度の向上と難読化された外部ライブラリのアップデートが今後の課題となる．

7. おわりに

本稿では，Android アプリケーションのユーザによる外部ライブラリアップデート手法を提案した．本提案では Android 端末上の Android アプリケーションに対し，外部ライブラリを外部サーバと連携してアップデートすることにより，ユーザが自身で Android アプリケーションに含まれる外部ライブラリの脆弱性を無くすことが可能である．既存の Android アプリケーションに対し，提案手法を適用し評価した結果，Android アプリケーションに含まれる外部ライブラリを特定できることを確認した．また smali ファイル置き換えによる，ライブラリアップデートが成功することを確認した．今後は，Android アプリケーションに含まれる外部ライブラリのバージョンが，最新のバージョンに比べ，古すぎる場合でも，提案手法が利用できる手法を検討する．また，難読化された外部ライブラリもアップデートできる手法を検討する．

参考文献

[1] News, T. H.: Facebook SDK Vulnerability Puts Millions of Smartphone Users' Accounts at Risk. `http://thehackernews.com/2014/07/facebook-sdk-vulnerability-puts.html`.

[2] Blog, D. D.: Security bug resolved in the Dropbox SDKs for Android. https://blogs.dropbox.com/developers/2015/03/security-bug-resolved-in-the-dropbox-sdks-for-android/.

[3] Database, V. N.: Apache Commons Collections Java library insecurely deserializes data. `http://www.kb.cert.org/vuls/id/576313`.

[4] Backes, M., Bugiel, S. and Derr, E.: Reliable third-party library detection in Android and its security applications, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 356–367 (2016).

[5] Grace, M. C., Zhou, W., Jiang, X. and Sadeghi, A.-R.: Unsafe exposure analysis of mobile in-app advertisements, *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pp. 101–112 (2012).

[6] Book, T., Pridgen, A. and Wallach, D. S.: Longitudinal analysis of android ad library permissions, *arXiv preprint arXiv:1303.0857* (2013).

[7] Chen, K., Liu, P. and Zhang, Y.: Achieving accuracy and scalability simultaneously in detecting application clones on android markets, *Proceedings of the 36th International Conference on Software Engineering*, pp. 175–186 (2014).

[8] Narayanan, A., Chen, L. and Chan, C. K.: Addetect: Automated detection of android ad libraries using semantic analysis, 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), pp. 1–6 (2014).

[9] Liu, B., Liu, B., Jin, H. and Govindan, R.: Efficient privilege de-escalation for ad libraries in mobile apps, *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 89–

103 (2015).
[10] Gibler, C., Stevens, R., Crussell, J., Chen, H., Zang, H. and Choi, H.: Adrob: Examining the landscape and impact of android application plagiarism, *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 431–444 (2013).
[11] Crussell, J., Gibler, C. and Chen, H.: Andarwin: Scalable detection of semantically similar android applications, *European Symposium on Research in Computer Security*, pp. 182–199 (2013).
[12] Wang, H., Guo, Y., Ma, Z. and Chen, X.: Wukong: A scalable and accurate two-phase approach to android app clone detection, *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pp. 71–82 (2015).
[13] Ma, Z., Wang, H., Guo, Y. and Chen, X.: Libradar: Fast and accurate detection of third-party libraries in android apps, *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 653–656 (2016).
[14] Mindorks: Android-HotFix. `https://github.com/MindorksOpenSource/Android-HotFix`.
[15] Simonin, D.: Fossdroid. `https://fossdroid.com/`.
[16] iText Software: ITEXT Developers. `http://developers.itextpdf.com/`.
[17] Ericksen, J.: Parceler. `https://github.com/johncarl81/parceler`.
[18] Square: Retrofit. `http://square.github.io/retrofit/`.
[19] Square: OkHttp. `http://square.github.io/okhttp/`.
[20] QOS.ch: SLF4J Simple Logging Facade for Java. `https://www.slf4j.org/`.