Androidアプリに含まれるコーディングエラーに関する考察

河村 辰也† 小柳 和子†

†情報セキュリティ大学院大学
221-0835 神奈川県横浜市神奈川区鶴屋町 2-14-1
{mgs115102, oyanagi}@iisec.ac.jp

あらまし 世界的に普及しているモバイル向けOSであるAndroidにおいて，マルウエアの脅威と同時に開発者の知識不足が原因として発生するアプリの脆弱性も多数報告されている．最近Android向けのセキュアコーディングガイドラインが出始めている．本研究では，ガイドラインの内容を自動チェックできるようにFindBugsのプラグインを開発している．その前段階として市場に出回っているアプリを解析し，FindBugsはAndroidアプリをJavaアプリと同様に解析できること，解析に難読化の影響は少ないこと，少数のアプリを解析した場合と多数のアプリを解析した場合でコーディングエラーの出現傾向が似ていることがわかった．

A Study on the Coding Errors contained in Android apps

Tatsuya KAWAMURA† Kazuko OYANAGI†

†Institute of Information Security
2-14-1 Tsuruya-cho, Kanagawa-ward, Yokohama 221-0835, JAPAN
{mgs115102, oyanagi}@iisec.ac.jp

Abstract Recently Android has become popular. A number of vulnerabilities have been reported in Android applications due to the lack of developers' knowledge. In other hands, Secure Coding Guidelines for Android has been enhanced. In this study, we develop plugin software of FindBugs. It will automatically check the contents of the guidelines. First, we analyzed application of the market before the plugin development. And we find that Android applications can be analyzed in FindBugs as well as Java applications, obfuscation has little impact on the analysis, and the trend of coding errors in the small exam is similar to those in the large exam.

1 はじめに

近年，世界的に Android が急速に普及しており，その広い普及を狙うかのように様々なマルウエアが出現している．Google 社が Android のアプリマーケットである Google Play Store に公開されているアプリを自動でチェックする Bounser[1]を公開したあとも「the Movie」系アプリ[2]で数万件から数十万件の個人情報が流出し，「TigerBot」という通話盗聴可能なマルウエア[3]が出現するなど，Android に対する深刻な脅威は減少しそうにないのが現状である．

その一方で, IPA の報告[4]によると Android

アプリで脆弱性報告されるものの 7 割以上がアクセス制限の不備などが原因のものである．これら脆弱性の原因は Android の特徴的な機能である，アクティビティやコンテントプロバイダ，Intent などのアクセス制御機能の設定不備から生じており，開発者へ Android 特有の機能，設定内容がしっかりと知られていないために発生したものと推測できる．

そこで本研究では，アプリ開発者の視点にたち，最近充実してきた Android のセキュアコーディングガイドラインの内容を自動でチェックできる FindBugs のプラグインを開発している．本稿では，その過程で行った，FindBugs が Android アプリでも使用可能か，難読化がかかっていても解析できるかの確認と，アプリ数を増やして出現するバグの傾向を調査したので報告する．

2 関連研究

Android は国内外で盛んに研究がおこなわれており，非常に進歩の早い分野である．アプリを解析するためのデコンパイラを開発する研究や，OS レベルで不正を防ごうとする研究，アプリを解析する研究などがある．

Enck らによる研究[5]は, ded という Android アプリ用のデコンパイラを開発し，1100 個の有名アプリをデコンパイルして開発したツールの精度を確かめている．このデコンパイラは本研究の実験でも使用している．

川端らによる研究[6]は，危険を伴うパーミッションに関連する API に割り込み処理を追加実装し，その API が動作する際にユーザに許可を求めるようにし，市販されている端末上で動作確認し，常用に耐える速度で動作することがわかったとしている．

Chin らの研究[7]では, すでに市場に出回っている Android アプリを解析する Web サービスを開発している．このサービスは APK ファイルを用意されたフォームからアップロードすると，解析プログラムが動作し，アクティビティが乗っ取られる可能性があるかなどを表示してくれる. なおこの Web サービスの ComDroid[8]は一般公開されており, 誰でも自由に利用することができるようになっている．

3 アプリ解析実験

3.1 実験目的

前章の関連研究で述べたように，Android OS やそのアプリについて様々な研究がなされているが，一方で IPA に報告[4]される Android アプリの脆弱性情報は 7 割がファイルアクセス制御不備という，Android に対する知識がしっかりとあれば防げた問題であると考えられる．最近，Android のセキュアコーディングガイド[9][10]は出始めているので, 開発環境に追加する形でこれらのガイドを自動チェックする仕組みを開発し，開発者の Android に対する知識不足からくる脆弱性の作りこみを少しでも軽減しようと考えている．

Android の静的解析ツールには Android Lint[11]があるが, このツールはセキュリティ重視と言うより，レイアウトやパフォーマンス，ユーザビリティのチェックに重きをおいている．さらに，解析ルールを独自で作成できるようになっておらず，追加が難しい．Android に対応しているわけではないが, Java のバイトコード静的解析ツールである FindBgus[12]は独自の解析ルールをプラグインという形で開発できるようになっている．そこで今回は FindBugs を用いることにした．

実験の目的は，FindBugs が Android アプリに対して解析解析を実行可能かと難読化の影響を調査することである．まず，15 個のアプリを対象として実験を行い，次に，1350 個のアプリを対象として，対象数を広げた場合の難読化の影響と，検出されるバグの傾向を調査した．

3.2　実験環境・方法

表 1 は，実験で使用する Android アプリを収集する環境を示す．ブラウザの Google Chrome に Android Play ストアから無料アプリを APK 形式でダウンロードすることができるアドオンの APK Downloader[13]をインストールし，無料人気アプリ上位 15 個と，全 27 カテゴリから無料アプリ上位 50 個, 計 1350 個をそれぞれ収集した．

表 2 は，収集した Android アプリを解析する環境を示す．収集以降の解析作業はすべて Mac OS X 上で行った．デコンパイラに先の関連研究で述べた ded を使用し，静的解析ツールに FindBugs を使用した．

表 1. アプリ収集環境

ブラウザ	Google Chrome
収集用アドオン	APK Downloader 1.2.1
OS	Windows XP

表 2. アプリ解析環境

デコンパイラ	ded 0.7.1
静的解析ツール	FindBugs 2.0.0
解析マシン	MacPro 2.8GHz Quad Core Intel Xeon (x2)
解析 OS	MacOSX 10.7.4

図 1 にアプリ解析の手順を示す．まず，先のアプリ収集環境で集めたアプリを ded デコンパイラにて，class ファイルへデコンパイルする．次に FindBugs を用いてすべての class ファイルを解析し, XML 形式のレポートファイルを出力させる．ded によるデコンパイルから，FindBugs によって XML 形式のレポートファイルを出力させるまではシェルスクリプトを作成し，並列，自動化して処理を行った．

図 1. アプリ解析手順

3.3　実験結果と考察

解析の結果，15 個のアプリからは全部で 134 種類 14,553 個のコーディングエラーが検出された．図 2 に検出された回数の多かった上位 10 個の出現回数のグラフを示す．

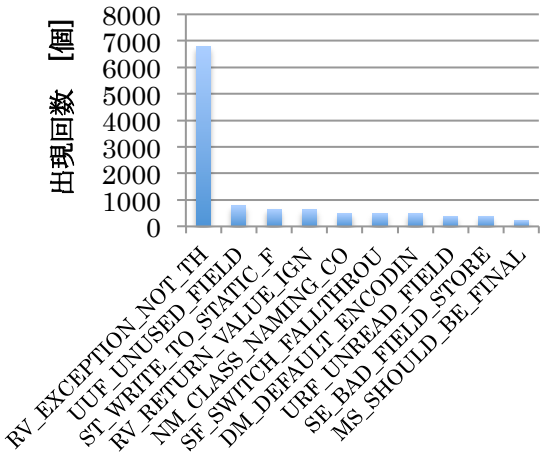

図 2. 15 アプリでの実験結果

1 位の RV_EXCEPTION_NOT_THROWN は例外を適切に処理していないことを警告するものである．

2 位の UUF_UNUSED_FIELD は未使用のフィールドが存在することを警告するものである．

3 位の ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD はインスタンスメソッドからスタティックへ書き込もうとしてい

ることを警告するものである．

4 位の RV_RETURN_VALUE_IGNORED はメソッドからの戻り値を破棄していることを警告するものである．

5 位の NM_CLASS_NAMING_CONVENTION は Java においてクラス名は大文字からはじめるべきであるが小文字から始まっていることを警告するもので，これは難読化でクラス名が”a”などの意味のない文字列に置き換えられているからであると考えられる．

6 位の SF_SWITCH_FALLTHROUGH は制御文である switch 文で分岐し，必要な処理を実行した後に break 文がない場合に発せられる．これはフォールスルーといい，他の条件の処理も意図せずに実行されてしまうという非常によく発生するバグにつながる．

7 位の DM_DEFAULT_ENCODING は文字エンコーディングの設定がされておらず，環境に設定されているデフォルトエンコーディングに依存した処理が行われるため，文字化けなどが発生する可能性がある場合に発せられる．

8 位の URF_UNREAD_FIELD は変数が宣言されているものの一度も読み出されないものがある場合に発せられる．これが出た場合は対象の変数の削除を検討するとよい．

9 位の SE_BAD_FIELD_STORE はシリアライズ出来ない値がシリアライズ可能と宣言されたクラスのインスタンスフィールドに格納された場合に発せられる．

10 位の MS_SHOULD_BE_FINAL は変更可能なスタティックフィールドが存在した場合に発せられる．フィールドに final 属性を付与し，パッケージプライベートにすることで悪意を持ったコードから書き換えられないようにすべきである．

これら実験結果より，Android アプリケーションは Java が開発言語であるので，Java の静的解析ツールはそのまま使用できることが分かった．また，難読化されているアプリケーションが多いが，難読化の影響で検出したであろうものは 5 位のもののみで，解析結果には大きく影響はないことも分かった．

1350 個のアプリからは全部で 217 種類 545,081 個のコーディングエラーが検出された．図 3 に検出された回数の多かった上位 10 個の出現回数のグラフを示す．なお，10 個中 7 個が 15 個のアプリの実験時に既出なので，説明は未出の 3 個についてのみ行う．

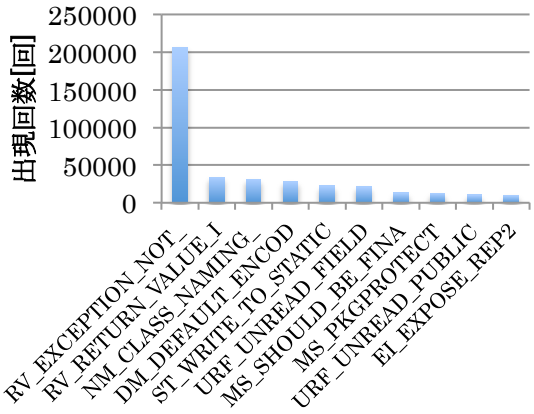

図 3. 1350 アプリでの実験結果

8 位の MS_PKGPROTECT は，外部から変更可能な static フィールドがあるときに発せられる．

9 位の URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD は，一度も読み出されない public または protected なフィールドが存在するときに発せられる．解析範囲外でも使用されていない場合は削除したほうが良い．

10 位の EI_EXPOSE_REP2 は外部の変更可能なオブジェクトを自分のクラス内に格納しているときに発せられる．

実験結果より，難読化の影響は上位 10 位のうちでは 3 位の NM_CLASS_NAMING_CONVENTION のみであり，15 個のアプリでの実験と同様解析結果には大きく影響は無いと判断した．バグの出現数の傾向もアプリ数を多くしても 1 位が非常に多く 2 位以下は 4 分の 1 以下と類似している．

表 3 は，15 アプリの解析結果と 1350 アプリの解析結果の出現件数のランクをまとめたものである．この表からわかることは，アプ

リの数が増えても，出現するコーディングエラーの種類は大きく変わらないということである．

今回の解析の限界として，どのコーディングエラーがアプリに対してどの程度の危険があるのかという観点では解析をおこなっていない. この解析を行うには FindBugs ではない静的解析ツールを使わなくてはいけないと考える．

表 3. コーディングエラーの出現頻度ランク

エラー名	15 アプリ	1350 アプリ
RV_EXCEPTION_NOT_THROWN	1	1
UUF_UNUSED_FIELD	2	–
ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD	3	5
RV_RETURN_VALUE_IGNORED	4	2
NM_CLASSNAMING_CONVENTION	5	3
SF_SWITCH_FALLTHROUGH	6	–
DM_DEFAULT_ENCODING	7	4
URF_UNREAD_FIELD	8	6
SD_BAD_FIELD_STORE	9	–
MS_SHOULD_BE_FINAL	10	7
MS_PKGPROTECT	–	8
URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD	–	9
EI_EXPOSE_REP2	–	10

4 Bugroid

前章では，Android アプリにおいて，FindBugs が問題無く使用出来る事を確認した．本章では現在実装中の FindBugs の Android プラグインである Bugroid について述べる．

最近，充実してきた Android のセキュアコーディングガイドは主に**表 4** にまとめたことについて扱っている．この内，FindBugs のプラグインである Bugroid とし実装を考えているのは Bugroid 欄にチェックが入っているものである．ファイルパスのハードコーディングなど，Android の API を使用せずにおこなっていることや，ログメッセージへの重要情報を書き込むことによる情報流出につながる可能性のあるコードの検出を始めとし，Intent 使用時のデータチェックで外部から不正なデータの注入に対してアプリが適切なチェックを行なっているかなどの確認をメインに実装していく．

Bugroid を使用すると，セキュアコーディングガイドラインをベースとした，チェックが出来るだけでなく，セキュリティに詳しくない開発者への教育にもなると考える. また，Bugroid は FindBugs の検出器プラグインという形で実装するので, 同時に FindBugs に標準で入っている強力な Java コーディング規約のチェックも同時に使用することができる．

類似のコードチェック機能を持つ Android Lint も一部 Bugroid で実装する機能と同じチェック機能を持っているが，このツールは現在，主に UI や翻訳，ユーザビリティのチェックに重きを置いているので，Bugroid と Android Lint は十分に共存できると考えている．

表 4. セキュアコーディングガイドの内容と Bugroid で実装するチェック項目と Android Lint のチェック項目の比較

内容	Bugroid	Android Lint
固有識別子の使用チェック	✔	–
ファイルパスのハードコーディング	✔	✔
Log への情報流出チェック	✔	–
Intent 使用時のデータチェック	✔	–
ファイルのアクセス制御チェック	✔	✔
外部記憶装置でのファイルの扱い	✔	–
データベースの保護	✔	–
パーミッションの組み合わせ	–	–
コンテントプロバイダの保護	–	–
Intent の保護	✔	–
有料アプリのライセンシング	–	–
難読化の設定	–	✔

5 まとめと今後

本稿では，FindBugs を用いて無料の Android アプリ 1350 個を解析し, コーディングエラーの出現傾向について調査し報告した．

コーディングエラーの出現傾向は解析アプリ数を増やしても大きく変化しなかった. したがって, Android 特有の問題点でも同じ傾向がでると考えられるので，今後は 4 章で述べた FindBugs のプラグイン Bugroid を早急に実装し, 先の実験で使用した 1350 個のアプリを用いて, プラグインの評価, 考察を行ってゆく．

Android は現在も開発が盛んに行なわれており，OS のバージョンアップ，開発ツール

のバージョンアップも早いので，開発中も定期的にチェック内容を見直し，最新の環境，新たな問題に対応できるように進めていく．

なお，Bugroid が使用出来るレベルに完成したら，オープンソースとして公開する予定である．

参考文献

[1] GoogleBlog, Android とセキュリティ http://googlejapan.blogspot.jp/2012/02/android.html [Accessed 2012.06.07]

[2] Gigazine, Android の「the Movie」アプリで数万件～数百万件の個人情報が大量流出した可能性あり http://gigazine.net/news/20120413-android-the-movie/ [Accessed 2012.06.13]

[3] Gigazine, Android に感染する新型マルウエア「TigerBot」登場，通話盗聴も可能に http://gigazine.net/news/20120412-tigerbot-new-android-malware/ [Accessed 2012.06.13]

[4] IPA テクニカルウォッチ，「Android アプリの脆弱性」に関するレポート ~簡易チェックリストで脆弱性を作り込みやすいポイントを確認しましょう~，http://www.ipa.go.jp/about/technicalwatch/20120613.html [Accessed 2012.06.13]

[5] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. “A Study of Android Application Security”, Proceedings of the 20th USENIX Security Symposium, August, 2011.

[6] 川端秀明，磯原隆将，竹森敬祐，窪田渉，可児潤也，上松晴信，西垣正勝，”Android OS における機能や情報へのアクセス制御機構の提案”，Computer Security Symposium 2011，pp.161-166，October 2011.

[7] Erika Chin, et al, ”Analyzing Inter-Application Communication in Android”, MobiSys ’11 Proceedings of the 9th international conference on Mobile system, applications, and services, pp.239-252, June 2011.

[8] COMDROID , http://www.comdroid.org [Accessed 2012.06.20]

[9] タオソフトウェア株式会社，”Android Security 安全なアプリケーションを作成するために”，インプレスジャパン，January 2012.

[10] JSSEC セキュアコーディンググループ，”Android アプリのセキュア設計・セキュアコーディングガイド”，June 2012.

[11] Android Lint http://tools.android.com/tips/lint [Accessed 2012.07.30]

[12] FindBugs http://findbugs.sourceforge.net/ [Accessed 2012.06.14]

[13] APK Downloader http://codekiem.com/2012/02/24/apk-downloader/ [Accessed 2012.07.25]