Android クローンアプリの大規模分析

石井 悠太 †　　渡邉 卓弥 †　　秋山 満昭 ‡　　森 達哉 †

† 早稲田大学 基幹理工学研究科　　‡NTT セキュアプラットフォーム研究所
169-8555　東京都新宿区大久保 3-4-1　　180-8585　東京都武蔵野市緑町 3-9-11

あらまし Android アプリはリパッケージングによる改変が容易であるため，オリジナル版開発者に無許可で開発されたリパッケージングアプリ (クローン) が広く流通している．クローンの多くが悪意のある目的で作成されている一方で，アプリ生成サービス等によって作成され，意図せず元のアプリに類似したアプリ (関連アプリ) も存在する．本研究は公式およびサードパーティーマーケットで収集した 130 万超の Android アプリを解析し，これらクローンと関連アプリの実態調査を目的とする．そのために類似アプリを抽出してクローンと関連アプリの分類する APPraiser フレームワークを開発した．分析の結果，公式マーケットでは類似アプリの 76%が関連アプリであったのに対し，サードパーティーマーケットでは類似アプリの 45%がクローンであったことや，公式マーケットからサードパーティーにクローンされたアプリの 80%が悪性アプリであることが判明した．

A Large-scale Analysis of Cloned Android Apps

Yuta Ishii†　　Takuya Watanabe†　　Mitsuaki Akiyama‡　　Tatsuya Mori†

†Waseda University
3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555
‡NTT Secure Platform Laboratories
3-9-11 Midoricho, Musashino-shi, Tokyo 180-8585

Abstract Since it is not hard to replicate an Android app, there are many cloned apps, which we call clones in this work. Generally, clones are generated for bad purposes by malicious parties. Besides clones, there are legitimate, similar apps, which we call relatives in this work. These relatives are not clones but are similar in nature, i.e., they are generated by the same app-building framework. Given these observations, this paper aims to answer the following two research questions: (RQ1) How can we distinguish between clone and relatives? (RQ2) What is the breakdown of clones and relatives in the official and third-party marketplaces? To answer RQ1, we developed a scalable framework called APPraiser that systematically extracts similar apps and classifies them into clones and relatives. To answer RQ2, we applied the APPraiser framework to the 1.3 millions of apps collected from official and third-party marketplaces. Our analysis revealed the following findings: In the official marketplace, 76% of similar apps were relatives while, in the third-party marketplace, 45% of similar apps were clones. The majority of relatives were apps developed by prolific developers in both marketplaces.

1　はじめに

Android は世界で広く利用されているモバイル端末向けのオープンソース OS である．スマートフォンをはじめとする多くのデバイスが Android を搭載している．全世界における Android スマートフォンの出荷台数は 2014 年には 1 億台を突破し [1]，Google Play 上のアプリ数は 2015 年 7 月時点で 160 万を超えている [2]．こうした数多くのデバイスで動作する Android アプリのうち，正規アプリを模倣したアプリが無視できない量存在する．例えば Zhou らは 6 つの異なるサードパーティーマーケットから収集した約 23,000 のアプリうち，5〜13%がリパッケージングされていたと報告している [3]．彼らはまた，収集した約 85,000 のアプリのうち 0.97〜2.7%が，正規アプリに悪性コードを追加した"piggybacked apps"であったとも報告している [4]．本研究ではこうしたリパッケージングアプリを総称して**クローン**と呼ぶ．実際に多くのクローンアプリが存在するという事実からもわかるように，Android アプリのリパッケージングは難しいことではない．こういった行為を容易に実現する多くのツールも存在している [5]．

先行研究が示すように，クローンの多くは悪意のある目的で作られている [3,4]．元のアプリには無かった広告モジュール挿入や広告ライブラリ顧客 ID の改変，個人情報を窃取する悪性コードの挿入などがそうした例である．このほか，有料アプリを違法にリパッケージング・改変する海賊版もクローンの一種である．これらのクローンの存在はエンドユーザだけでなく．アプリ開発者や著作権者，マーケット提供者にとっても有害である．

クローンとは別に，意図せず互いの外見や挙動が類似したアプリが存在する．本研究ではこのようなアプリを**関連アプリ**と呼ぶ．関連アプリはアプリ自動生成サービスによるものと，同一開発者によるものに大別することが出来る．これらは固定のテンプレートから生成されている可能性が高い．

クローンも関連アプリも，他のアプリと類似している点では本質的に同じである．しかし前者は有害であ

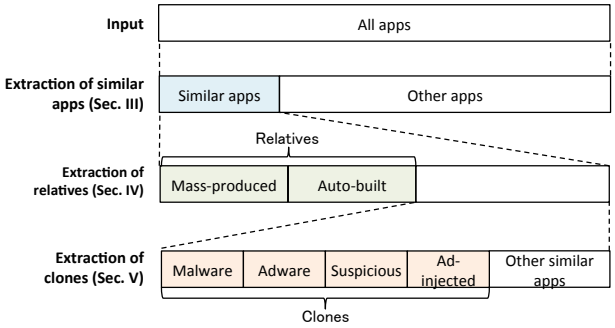


図 1: APPraiser フレームワークの概要図

りマーケットから取り除かれるべきであるから，両者を区別する必要がある．本研究の目的は，以下の課題を明らかにすることである．

RQ1: **いかにしてクローンと関連アプリを区別するか**?

RQ2: **公式およびサードパーティーマーケットにおけるクローンと関連アプリの実態はどのようなものか**?

RQ1 に答えるため，我々は *APPraiser* と名付けるフレームワークを開発した．*APPraiser* は自動で類似アプリを抽出し，それらをクローンと関連アプリ，そしてさらなるサブカテゴリに分類する．本フレームワークの要所は 3 つのステップからなる．はじめにアプリの外見により類似アプリを抽出し，次に開発者情報に基づいて関連アプリを抽出する．最後にコード差分やアンチウイルスソフトの分析結果によってクローンの分類を行う．**RQ2** に答えるため，我々は公式およびサードパーティーマーケットから収集した 130 万以上のアプリに本フレームワークを適用した．これら 2 つのマーケットのアプリを分析することで，マーケット内およびマーケット間のクローンと関連アプリの分析が可能となった．

第 7 章に述べる制限はあるが，*APPraiser* は 130 万を超えるアプリに対しても十分なスケーラビリティを発揮し，マーケットに流通するクローンの全体像をつかむことができた．分析により以下のことが明らかになった．公式マーケットでは類似アプリの 76%が関連アプリであったのに対し，サードパーティーマーケットでは類似アプリの 45%がクローンであった．また両マーケットにおいて関連アプリの大半が同一開発者によって大量生産されたものであった．また，公式マーケットからサードパーティーにクローンされたアプリのうち，80%が悪性であるということも判明した．

本論文の構成を以下に示す．第 2 章では *APPraiser* フレームワークの概要を述べる．第 3 章，4 章，5 章では，類似アプリの抽出，関連アプリの抽出，クローンの抽出についてそれぞれ説明する．第 6 章では，我々のデータセットに *APPraiser* フレームワークを適用した分析結果の主要な発見について述べる．第 7 章では *APPraiser* の制限事項と今後の課題について述べる．第 8 章では本研究に関連する研究を紹介し，最後に第 9 章をもって本研究のまとめとする．

2 APPraiser フレームワークの概要

本章では *APPraiser* フレームワークの目標と概要を説明する．本フレームワークの目標はアプリ群の中からクローンと関連アプリを抽出することである．ここで課題となるのが，大きなデータセットに対してもスケーラビリティを維持できるかということである．これらに対処するため，*APPraiser* フレームワークでは以下に説明する 3 つのステップを採用する．

図 1 は *APPraiser* フレームワークの概要図である．はじめに第 3 章で説明する手法を用いて，アプリの外見により類似アプリを抽出する．抽出された類似アプリは類似する各クラスタに分割される．*APPraiser* フレームワークはアプリのメタ情報に基づき，各クラスタの元アプリを定める．第 2 のステップでは，*APPraiser* フレームワークは関連アプリを抽出する．関連アプリには多くのアプリを生産している多作の開発者による大量生産系と，アプリ自動生成サービスによって作られた自動生成系が含まれる．これらの抽出方法の詳細については第 4 章で述べる．第 3 ステップでは，*APPraiser* フレームワークはクローンを抽出し，これらをマルウェア，アドウェア，疑わしいアプリ，広告挿入アプリという 4 つのカテゴリに分類する．これを行うため，我々はアンチウイルスソフトおよびコードの差分分析を行った．この詳細は第 5 章で述べる．

3 類似アプリの抽出

本章では，*APPraiser* フレームワークが類似アプリを抽出する方法を説明する．その基本的なアイディアは，アプリの外見情報を用いて類似度を算出することである．これは明確な意図を持ってアプリのクローンをつくるとき，その外見は似ている可能性が高いという予測に基づく．例えば悪性クローンの目的はユーザを惹きつけて正規アプリと偽って利用させることであるから，そこに外見を変更する理由はない．また関連アプリは，大半のアプリがお互いに共通するリソースを多く持っていることがわかった．したがって，類似アプリの多くはその元アプリと外見が類似していると推測できる．実際に文献 [6] や [7] では類似アプリの検出にリソースファイルを用いている．これら既存研究では類似アプリの検出自体は同様の手法をとっているが，クローンと関連アプリの区別を行ってはいないことに注意されたい．アプリの外見の分析にあたり，我々はアプリのパッケージファイルに含まれるリソースファイルを採用した．以降では，はじめに Android のアプリファイルから分析に用いる情報を抽出する方法を説明する．次に，これらの情報を用いて巨大なアプリ群から類似アプリを抽出する方法について述べる．さらに，抽出した類似アプリをどのようにクラスタに分けていくのかについて説明する．

3.1 APK ファイルの処理

Android アプリのパッケージファイルである APK には開発者証明書やマニフェストファイル，DEX ファイル，リソースや assets ファイル等が含まれる．このうち開発者証明書はアプリの開発者情報の抽出に利用できる．マニフェストファイルはパーミッションアクセスに代表される，アプリの主要な情報が書き込まれている．

このマニフェストファイルを分析することで，元アプリからどのパーミッションが追加あるいは削除されたのかを知ることができる．DEX ファイルは Android OS 上でアプリを実行する仮想マシンである Dalvik 用のバイトコードでできている．この DEX ファイルは smali [8] のようなツールを用いてディスアセンブルすることができる．こうして得られた smali コードを同様に分析することで，元アプリから追加/削除された API コールの分析が可能となる．リソースファイルや assets ファイルは主にアプリの外見部分を担う．これらは画像ファイルや音声ファイル，スクリーンのレイアウトを定義した XML ファイル等から構成される．次節ではこれらの情報をどのように扱うのかについて説明する．

3.2 外見の分析

先行研究 [6, 7] で類似アプリの検出に使っていたリソースファイルに加え，我々は `assets` と `lib` フォルダ内のファイルも対象に含めた．これらも外見に寄与するファイルといえる．

以下では，リソースファイルを使って類似アプリを抽出する手法を説明する．まずはじめに，それぞれのリソースファイルの MD5 ハッシュ値を計算する．次に，テキスト分類で広く用いられている DF-thresholding の手法を適用する [9]．これによって，数多くのアプリに共通して現れるような頻度の高いリソースを類似計算対象から除外する．本実験では経験的に $K = 100,000$ を閾値とし，リソースの出現頻度の上位 K 個を除外した．これは全リソースのおよそ 0.1%に相当する．

アプリ間の外見の類似度計算には Jaccard 係数を用いた．Jaccard 係数は 2 つの集合の類似度を示す指標のひとつである．アプリ x のもつリソースハッシュ値の集合を $\mathbf{R}(x)$ とすると，アプリ a とアプリ b の Jaccard 係数は $J(a,b) = \frac{|\mathbf{R}(a) \cap \mathbf{R}(b)|}{|\mathbf{R}(a) \cup \mathbf{R}(b)|}$ と計算される．この値域は 0 から 1 までである．もし両者のアプリに共通するリソースが全くない場合は Jaccard 係数の値は 0 となり，逆にすべてのリソースが一致していた場合は値は 1 となる．リソースファイルの抽出にあたり，Android のリバースエンジニアリングツールである Androguard [10] を用いた．

図 2 に示すのは Jaccard 係数とそのときの類似判定されたアプリのペア数との累積分布関数である．ここですべてのペア数は $n(n-1)/2$ であり，実際に類似判定されたアプリのペア数よりもはるかに大きいことに注意されたい．図をみると，ほとんどのペアの Jaccard 係数が 0 となっていることがわかる．実際 99.98%以上のペアの Jaccard 係数が 0 である．

2 つのアプリ間の類似性を決定する閾値として，我々は Jaccard 係数の値が 0.6 のときを採用した．つまりアプリ a とアプリ b の類似度 $J(a,b)$ がこの値以上であったとき，これらを類似アプリとして抽出する．分析にあたり，この閾値は結果にさほど大きな影響を及ぼさないことを確認した．すなわち，閾値が 0.5 や 0.7 の場合でも，結果に大きな違いはみられなかった．

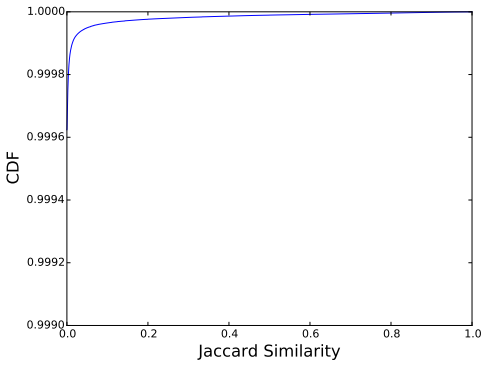


図 2: Jaccard 係数とアプリのペア数との累積分布関数

Algorithm 1: Jaccard 係数の計算アルゴリズム

```
1  c(x, y) = 0                          /* a counter of a tuple (x, y) */
2  S = ∅                                /* a set to check entrance */
3  T = ∅                                /* will be used in Algorithm 2 */
4  U = ∅                                /* will be used in Algorithm 2 */
5  for ∀a ∈ A do
6      for ∀r ∈ R(a) do
7          for ∀b ∈ I(r) do
8              c(a, b) ← c(a, b) + 1
9              if (a, b) ∉ S then
10                 add (a, b) into S
11 for (a, b) ∈ S do
12     J(a, b) = c(a, b)/ (|R(a)| + |R(b)| − c(a, b))
13     if J(a, b) ≥ 0.6 then
14         if (a, b) ∉ T then
15             add (a, b) into T
16         if a ∉ U then
17             add a into U
```

3.3 高速な Jaccard 係数計算アルゴリズム

単純にアプリの全ペアについて Jaccard 係数を計算しようとすると，1 対 1 の比較が必要となる．この方法は計算量が $O(n^2)$ であり，スケーラビリティに欠けるのは言うまでもない．今回の場合，$n \approx 1.3 \times 10^6$ である．そこで我々はデータの持つ疎性を利用した．つまりペアの多くは共通するリソースを持っておらず，そのようなペアの Jaccard 係数は 0 となる．ここで全アプリの集合を $\mathbf{A}$ とし，あるリソース r を持つアプリの集合を $\mathbf{I}(r)$ と置く．全アプリのペアに対する Jaccard 係数を計算するアルゴリズムをアルゴリズム 1 に示す．ここで $(a,b) \notin S$ ならば Jaccard 係数は $J(a,b) = 0$ である．

以下では本アルゴリズムの計算量を考える．$\mathbf{R}(\cdot)$ は $n = |\mathbf{A}|$ に依存しないため，オーダーは $O(|\mathbf{A}|\langle\mathbf{I}\rangle) = O(n\langle\mathbf{I}\rangle)$ と表せる．ここで $\langle\mathbf{I}\rangle$ は $|\mathbf{I}(r)|$ の期待値であり，$\mathbf{R}$ をすべてのリソースの集合としたとき，$\langle\mathbf{I}\rangle = \frac{1}{|\mathbf{R}|}\sum_{r\in\mathbf{R}} |\mathbf{I}(r)|$ とあらわせる．最悪の場合はすべての r について $|\mathbf{I}(r)| = n$ のときであり，計算量は $O(n^2)$ となる．これはすべてのアプリが同一のリソースを持つときであるが，当然これは現実的ではない．実際にはデータの疎性のおかげで，大半のリソースについて $|\mathbf{I}(r)| = 1$ が成り立つ．今回用いた $n = O(10^6)$ のデータセットでは $\langle\mathbf{I}\rangle = 1.72$ であった．したがって，このアルゴリズムはデータの疎性が担保されている場合に $O(n)$ で動作する．

Algorithm 2: 貪欲なクラスタリングアルゴリズム

```
1 G(x) = ∅                          /* a set of items in cluster x */
2 while U ≠ ∅ do
3     x = random(U)
4     for ∀y such that (x, y) ∈ T do
5         add y into G(x)
6         remove y from U
7     remove x from U
```

3.4 クラスタリングと元アプリの決定

前節では Jaccard 係数が 0.6 以上の類似アプリペア **T** を抽出した．続いてアルゴリズム 2 に示す貪欲法を用いて，類似アプリのクラスタを形成する．random(**X**) は集合 **X** からランダムに要素を取り出す関数である．なお **T** と **U** はアルゴリズム 1 ですでに計算済みである．ゆえにこのアルゴリズム 2 は軽量に動作する．

こうして得られたクラスタは常に最適なものになるとはいえない．しかし我々の研究目的は世に流通している類似アプリの実態を解明することであるから，今回は精度よりもスケーラビリティを重視した．この課題については第 7 章でも議論する．

最後にそれぞれのクラスタについて，クローンや関連アプリに対する元アプリを定義する．公式マーケットでは，各クラスタの中でアプリのダウンロード数が最多のものを元アプリとした．サードパーティーマーケットではマーケットにおけるアプリの ID を用いた．これには今回対象としたマーケットのアプリ ID が単純にインクリメントされているという特性を利用した．つまり，各クラスタで最小の ID をもつものがオリジナルである可能性が高いため，これを元アプリとした．

実際の元アプリがデータに含まれていなかった場合，この方法は有効でないということには注意しなければならない．すなわちクラスタ要素のアプリがすべて関連アプリかクローンのときは元アプリの判定はうまくいかないことになる．この問題は第 7 章でも述べる．

4 関連アプリの抽出

本章では *APPraiser* フレームワークが関連アプリを抽出する方法を説明する．これは同一の開発者あるいはアプリ自動生成サービスによって作られたものであることを示すフィンガープリントを適用することで実現する．本研究において，同一人物によって開発されたアプリはクローンと呼ぶべきではない．我々の想定するクローンは元アプリの開発者とは無関係の第三者に作られたアプリのことである．以降では各カテゴリの詳細やそれらの抽出方法について述べていく．

4.1 大量生産アプリ

少数の開発者が大量のアプリを開発しているという事例が報告されている [11]．そのような開発者によるアプリはお互いに類似する傾向にある．そのような開発者は短期間で大量のアプリを開発するため，共通のリソースを含んだテンプレートを使用しているものと推測する．また外注でアプリ制作を請け負うような企業も同一のテンプレートを使用している可能性がある．

表 1: 自動生成サービスとフィンガープリントの一覧

サービス名	フィンガープリント
Andromo	`andromo`
Appery.io	`appery`
appexpress	`appexpress`
AppMachine	`artistapp`
Apps Bar	`appsbar`
AppsBuilder	`appsbuilder`
Appy Pie	`appypie`
Bizness Apps	`app_***.layout`
como	`.conduit.`
GoodBarber	`goodbarber`
iBuild APP	`appbuilder`
MIT App Inventor	`appinventor`
ReverbNation	`reverbnation`
vBulletin Mobile Suite	`vbulletin`

これらのテンプレートやフレームワークを利用して作られたアプリはお互いに類似性をもつ．こうしたアプリを**大量生産アプリ**と呼ぼう．

開発者に関する情報は，開発者証明書と開発者名の 2 つから得ることができる．我々はまず APK ファイルから開発者証明書を抽出し，そこから公開鍵を取り出した．また，ある開発者が異なるアプリに別々の秘密鍵・公開鍵のペアを用いる可能性を考慮し，証明書のうち証明対象者の組織名 (O) と住所 (L) の組をもってフィンガープリントとした．さらに，アプリ開発専門会社のような開発者はそれぞれのアプリに対して別々の証明書を用いることもありうる．そこで我々は，マーケットから得られるメタ情報のうち開発者名もフィンガープリントとして用いた．

以上をまとめると，大量生産アプリ抽出のために開発者証明書とアプリ開発者名を利用した．そして各クラスタに少なくとも 2 つ以上の同一の公開鍵や開発者名を持つアプリが存在したとき，それらを大量生産アプリとして抽出する．

4.2 自動生成アプリ

クラウドベースのアプリ開発サービスがいくつか存在する [12, 13]．これらのサービスではコードを記述せずに Web 上のインターフェースのみでモバイルアプリを作成することができる．本研究では，これらのサービスによって作られたアプリを**自動生成アプリ**と呼ぶ．自動生成アプリは多くの不要なパーミッションや API を要求する傾向があることがわかっている [11]．また，実際には使用されていない共通のリソースを持っていることも多い．したがって自動生成アプリは異なる開発者が作ったものでも，お互いのリソースやコードは似通っている場合が多くなる．アプリパッケージ名の文字列の出現頻度を分析し，そのようなサービスの一覧を取得した．表 1 はそのようなサービス名と，それを抽出に利用したパッケージ名の部分文字列である．この表 1 を用いて自動生成アプリの抽出を行った．

ただしこの手法には明らかな制限がある．例えば自由にパッケージ名を付けられるようなアプリ生成サービスにはこの手法は通用しない．現時点で人気のあるサービスに対してはうまく抽出できたと考えられるが，上述のようなサービスへの対応は今後の課題とする．そうしたアプリは，パッケージ名以外にも何か共通の要素を持っていると期待する．

5　クローンの抽出と分類

本章では，残った類似アプリからクローンにあたるものを *APPraiser* フレームワークによって抽出し，それらを 4 つのカテゴリに分類する方法について説明する．分類するカテゴリはそれぞれマルウェア，アドウェア，疑わしいアプリ，広告挿入アプリである．

5.1　マルウェアとアドウェアの抽出

はじめにマルウェアやアドウェアのような悪質なクローンを抽出する．もしアプリ *B* が正規アプリ *A* のクローンであり，かつマルウェアやアドウェアだと検出された場合，アプリ *B* をアプリ *A* の悪質なクローンであるとみなす．この仮定に基づき，はじめに類似アプリがマルウェアやアドウェアかどうかを確かめる．ここで元アプリが正規のもの，すなわちマルウェアやアドウェアでないときに限りこの操作を行った．

本研究はマルウェアやアドウェアを検出する新たな手法の提案が目的ではないため，今回は単純に VirusTotal [14] を用いた．これは 60 以上の商用アンチウイルスソフトをまとめて検査できるオンラインサービスである．はじめに，残ったすべての類似アプリを VirusTotal にかける．あるアプリが少なくとも 1 つのベンダーによってマルウェアと判定された場合，そのアプリは悪性なクローン (マルウェア) であるとみなす．マルウェアではないが，少なくとも 1 つのベンダーによってアドウェアだと判定された場合，そのアプリは悪質なクローン (アドウェア) であるとみなす．

当然 VirusTotal の結果には誤検知や検出漏れが含まれるうえ，検出されたマルウェアやアドウェアが実際に元アプリをリパッケージングしたものなのかどうかまでは保証できない．この手法の有効性を確かめるため，ランダムサンプリングしたいくつかのアプリを手動で解析した．さらに上記の方法で検知しきれなかった潜在的なマルウェアやアドウェアを見つけるため，以降に示す 2 つのカテゴリを導入した．

5.2　疑わしいアプリ/広告挿入アプリの抽出

マルウェア/アドウェアに続いて 2 つのカテゴリを抽出する．**疑わしいアプリ**と**広告挿入アプリ**である．これらはそれぞれ前節で検出されなかったマルウェアやアドウェアを捕捉するためである．VirusTotal 適用後，コードの静的解析を行う．*APPraiser* フレームワークでは，パーミッションとそれに紐づく API コール，広告ライブラリの使用する FQDN を抽出し分析する．これらの特徴量はマニフェストファイルやディスアセンブルした DEX ファイルから抽出する．次に元アプリ *A* とその類似アプリ *B* について，抽出したそれぞれが持つ特徴量の差を分析する．

表 2 にアプリ *A* に追加されうる危険なパーミッション一覧を記す．アプリ *B* が表 2 のパーミッションを少なくとも 1 つ持ち，かつアプリ *A* がそのパーミッションを持たないとき，アプリ *B* を**疑わしいアプリ**に分類する．同様に API について，アプリ *B* が表 2 に含まれるパーミッションに紐づく API を少なくとも 1 つ持ち，かつ元アプリ *A* がその API を持たないとき，アプリ *B* を**疑わしいアプリ**に分類する．

表 2: 危険なパーミッションの一覧

Permissions	
ACCESS_FINE_LOCATION	SEND_SMS
ACCESS_COARSE_LOCATION	READ_SMS
ACCESS_LOCATION_EXTRA_COMMANDS	RECEIVE_SMS
READ_LOGS	WRITE_MEDIA_STORAGE
INSTALL_SHORTCUT	RESTART_PACKAGES
SYSTEM_ALERT_WINDOW	INSTALL_PACKAGES
SYSTEM_OVERLAY_WINDOW	ACCESS_WIFI_STATE
RECEIVE_BOOT_COMPLETED	DISABLE_KEYGUARD
CHANGE_NETWORK_STATE	READ_CONTACTS
DOWNLOAD_WITHOUT_NOTIFICATION	READ_PHONE_STATE
MOUNT_UNMOUNT_FILESYSTEMS	

表 3: 分析に用いたデータの内訳

マーケット	# 検体数	収集期間
Google Play	1,296,537	2014 年 10 月
Anzhi	74,185	2013 年 11 月 – 2014 年 4 月
合計	1,370,722	–

B が新たに広告を追加したかどうかも同様に分析した．以降では広告ライブラリの用いる FQDN を ad-FQDN とあらわす．分析のため，広告ライブラリが発生させる通信をブロックするために作られた ad-FQDN のリストを活用した．まずはじめに，AdAway [15] のような著名な広告ブロックサイトから ad-FQDN を収集した．次にそのリストから `schema.android.com` のような明らかに ad-FQDN でないものを除外した．最終的に用いた ad-FQDN の数は 1,027 である．最後にアプリをディスアセンブルしたコードを解析し，ad-FQDN のリストと照合する．もしもアプリ *B* がアプリ *A* に含まれない ad-FQDN を少なくとも 1 つ持つとき，アプリ *B* を**広告挿入アプリ**に分類する．

6　分析

本章では，大量の Android アプリの分析から得られた主な知見を述べる．まずはじめに我々が分析に用いたデータについて説明する．そしてそのデータセットに対して *APPraiser* フレームワークを適用することで，**RQ2** を解き明かす．最後にいくつかの APK をランダムサンプリングによって分析し，実験の評価を行う．

6.1　データ

本研究では公式マーケット [16] とサードパーティーマーケット [17] の両方から無料の Android アプリを収集した．同じアプリに対して複数のバージョンを収集した場合は最新バージョンを対象とした．また，何らかの理由で破損していたファイルも除外した．データの内訳を表 3 に示す．

これらのデータを用いることで，公式およびサードパーティーマーケット間の質的な差を分析することができる．公式マーケットでは Bouncer [18] と呼ばれる独自の防御システムが導入されていることが知られている．したがって先行研究にもあるように，公式マーケットはサードパーティーマーケットと比較して悪性アプリが少ない傾向にある [3]．一方で最も人口の多い国である中国では，Google Play 公式マーケットが利用できない．したがって公式マーケット上で人気のあるア

表 4: 類似アプリの数と割合

	Google Play	Anzhi
類似アプリ	191,112 (14.7%)	29,991 (40.4%)

表 5: 類似アプリの内訳

	Google Play	Anzhi
関連アプリ	145,038 (75.9%)	14,156 (47.2%)
クローン	18,447 (9.7%)	13,429 (44.8%)
不明	27,627 (14.4%)	2,406 (8.0%)

表 6: 関連アプリの内訳

	Google Play	Anzhi
大量生産アプリ	132,423 (91.3%)	14,154 (100.0%)
自動生成アプリ	12,615 (8.7%)	2 (0.0%)

プリを楽しみたい人が，そのアプリのクローンをサードパーティーマーケットでリリースする動機があり得る．本研究では，類似アプリという観点から両マーケットの性質の違いについて分析を試みる．

6.2 アプリの分類とそれらの性質

RQ2 を明らかにするため，*APPraiser* フレームワークを用いたアプリの抽出・分類結果を報告する．まず表 4 に両マーケット類似判定されたアプリの数と割合を示す．我々の予想通り，サードパーティーマーケットにおける類似アプリの割合は公式マーケットのそれと比べてかなり高くなった．公式マーケットにおいても無視できない数のアプリが類似アプリに分類された．

続いて，類似アプリの内訳を表 5 に示す．ここから Google Play の関連アプリの割合が極めて高いことがわかる．この結果は，リソースによって検出された類似アプリの多くはクローンではなく関連アプリであったということを示している．また Anzhi におけるクローンの割合が，Google Play でのそれに比べてはるかに高いということも注目に値する．以降では，これらのカテゴリのさらなる分析結果をみていく．

表 6 に示すのは関連アプリの内訳である．これをみると，**関連アプリ**の大部分は**大量生産アプリ**であるということがわかる．つまり，一部の開発者がおそらく同一のテンプレートをもちいて多くのアプリを開発していることがわかる．本研究ではサードパーティーマーケットにおける著名なアプリ生成サービスを発見することができなかった．その結果，サードパーティーマーケットの**自動生成アプリ**の数はほとんど 0 に近い．サードパーティーマーケットにおけるこうしたサービスを検出する手法については今後の課題とする．

図 3 にクローンの分類結果の内訳を示す．ここでは元アプリは含めていない．クロスマーケットでは，クローンの元アプリは公式マーケットのものという想定で分析を行った．公式マーケットでは約半数のクローンが**広告挿入アプリ**に分類されたのに対し，マルウェアの割合はわずか 10%ほどだった．サードパーティーマーケットと比較するとマルウェアの割合はかなり低い．これは公式マーケットで運用されている防御システム Bouncer によるものが大きいと考える．また，約 80%のクローンが商用のアンチウイルスソフトに検出されなかったということも分かった．したがってコードの静的解析の我々の手法は，潜在的なマルウェアやアドウェアを発見するのに有効だったことがわかる．サードパーティーマーケットをみると，クローンの 60%がマルウェアである．クロスマーケットではこの割合は 80%である．このことは，サードパーティーマーケットに存在する悪性なクローンの大部分が，公式マーケットのアプリを元にしているということを示している．

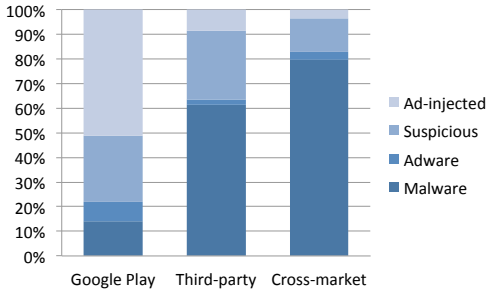


図 3: クローンの内訳

表 7: 元アプリのダウンロード数の統計

	平均値	中央値
全体	37,439	50 ~ 100
クローン	116,317	100 ~ 500

アプリカテゴリごとのクローンの作られやすさついて調べた．図 4 は公式マーケットでのカテゴリごとのアプリ数の分布である．多くのカテゴリで，すべてのアプリ，類似アプリ，クローンアプリの分布は似通っているが，ことゲームアプリに関してはクローンの割合が高いことがわかる．これは，クローンの作者は人気アプリをリパッケージングしたがるからだろうと推測する．表 7 からもこのことが伺える．これはクローンを作られた元アプリのダウンロード数の平均値と中央値を全体と比較したものである．いずれもクローンを作られたアプリのほうが値が大きい．ここで，Google Play のダウンロード数は 0 ~ 10, 10 ~ 50, 50 ~ 100 のように離散的な値となっている．上記の結果から，クローンはユーザを惹きつけるため，より人気のあるアプリをそのターゲットとするということが言える．

6.3 抽出・分類したクローンの評価

関連アプリの分類にはそれに固有のフィンガープリントを用いたので分類精度は高いと見込める一方，ク

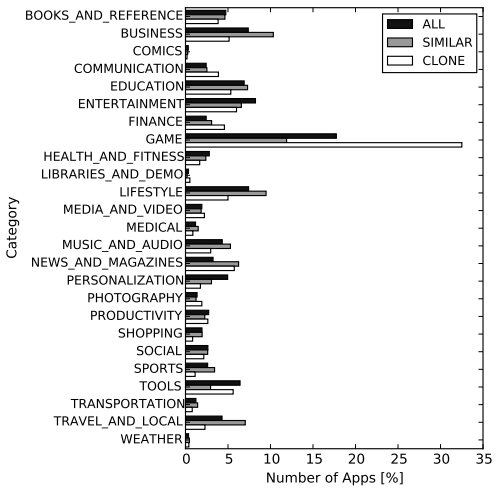


図 4: カテゴリごとのアプリの分布 (Google Play)

ローンに関しては分類の妥当性を評価する必要がある．各カテゴリからランダムサンプリングされた数十のアプリを静的・動的に手動で解析することで，クローンの分類精度を評価した．以降では各カテゴリについていくつかの例を示し，そこから得た知見を報告する．

はじめに図 5(a) で示すのはクロスマーケットにおける**マルウェア**のクローンである．クローンには元アプリには存在しない大きな広告が表示されていることが分かる．図 5(b) の例では，元アプリのダウンロード数は 10,000 ~ 50,000 の一方，クローンは 50 ~ 100 であった．画面をみるとわかるが，クローンアプリは起動後すぐにエラーで終了してしまった．両アプリともに 2015 年 7 月時点でマーケット上に残っている．

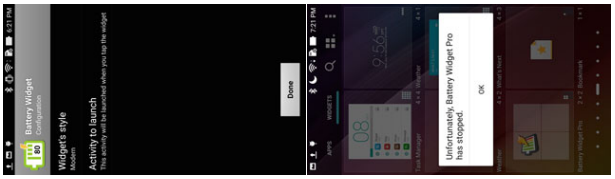
アドウェアの例を図 5(c) に示す．元アプリとはアプリアイコンや使用画像の一部はことなるものの，アプリ全体の構造は同一であった．クローンはすでにマーケットから削除されている．

疑わしいアプリでは，図 5(d) に示したものは見た目は同一だが，クローンでは `RECEIVE_BOOT_COMPLETED`, `READ_PHONE_STATE` 等のパーミッションが新たに追加されていた．API やサービスもそれぞれ `getDeviceId()` や `PushMessageService` 等が追加されていた．このクローンもすでにマーケットには存在しない．

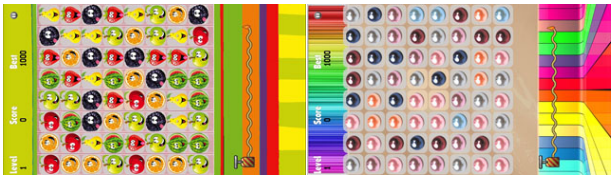
図 5(e) は**広告挿入アプリ**の例である．クローンには元アプリにはない広告が追加されているが，アンチウイルスソフトではアドウェアとしては検知されなかった．このクローンもマーケットから削除されている．

(a) マルウェア (クロスマーケット)

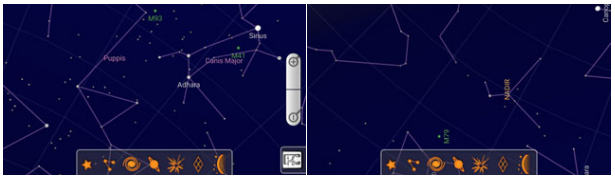
(b) マルウェア (Google Play)



(c) アドウェア (Google Play)



(d) 疑わしいアプリ (Google Play)



(e) 広告挿入アプリ (Google Play)

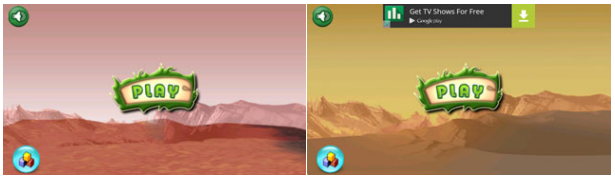


図 5: 元アプリ (左) とそのクローン (右)

7 議論

本章では *APPraiser* フレームワークのいくつかの制限事項について議論し，今後の展望について述べる．まず本研究では大量のデータを効率的に扱うため，類似アプリ検出にシンプルなアルゴリズムを採用した．したがって，生成されたクラスタが最適である保証は必ずしもない．そこで，その他のスケーラブルなクラスタリングアルゴリズムを適用し，結果の差異を観察することは今後の課題とする．第二の制限は解析対象アプリがすべて無料版であり，有料アプリが含まれていない点である．したがって有料アプリの海賊版に相当するクローンを発見することはできなかった．クローンアプリの全貌を解き明かすためには，これら有料アプリの分析も今後は視野に入れていく必要がある．またすでに述べたが，マルウェアやアドウェアの検出手段にアンチウイルスソフトを利用した．しかしアンチウイルスソフトには誤検知がつきものである．サードパーティーマーケットにおけるマルウェアでは，SecAPK [19] と呼ばれるサービスを用いて難読化が施されているケースが少なくなかった．これはバイトコードを難読化することでリバースエンジニアリングの防止を図るものである．今回のデータセットでは，VirusTotal によってこれらすべてが悪性と判定された．正規の元アプリを第三者がリパッケージングする際に，こうした難読化を施すのには何か理由があるはずだ．したがって，そのようなアプリは悪質なクローンと言って良いと考える．

8 関連研究

Android の**類似アプリ**を検出・分析する過去の研究は，コードベースによるものとリソースベースによるものに大別される．ここではそれぞれについての概要を述べ，本研究との違いを議論する．

8.1 コードベースの手法

DroidMOSS [3] はリパッケージングアプリを検出するフレームワークで，ディスアセンブルされたコードのオペコードを利用した．しかし各ペア同士の計算が必要なため，この計算時間は $O(n^2)$ となる．DNADroid [20] はクローンアプリを検出するフレームワークである．アプリ同士のプログラム依存グラフ (PDG) を展開し比較を行っている．これも計算時間は $O(n^2)$ となる．PiggyApp [4] フレームワークは “piggybacked app” を検出する．“piggybacked app” とは元アプリに対して悪性なコードを追加しリパッケージングしたアプリのことである．PiggyApp ではアプリをメインモジュールとそうでないモジュールに分割してコード分析を行う．メイ

ンモジュールを効率的に分析することで，$O(n \log n)$ の計算時間を実現した．

一般的にコードベースの計算コストは高くつく．例えば，確かに PiggyApp は $O(n \log n)$ と主張しているものの，モジュール分割処理のコストは無視できないはずだ．こうした計算コストのため，どの手法でも分析したアプリの数には限りがある．具体的に，DroidMOSS は $n = 68,817$，DNADroid は $n = 75,000$，PiggyApp は $n = 84,767$ である．コードによる手法のもう一つの欠点に，難読化や暗号化への対処の難しさがある．これらと比較して，リソースベースの手法は低コストかつ難読化・暗号化の影響を受けないという利点がある．次節ではこの手法について取り上げる．

8.2 リソースベースの手法

Viennot ら [7] の開発した PlayDrone と呼ばれるシステムは，公式の Google Play Store 上のアプリを効率的に収集するクローラである．彼らは PlayDrone によって収集された約 100 万の検体を用いて，類似アプリの分析を含む様々な分析を行った．互いに類似するアプリを検出するためにリソースを特徴として採用している．分析の結果，約 25%のアプリがクローンを含む様々な理由で他アプリと類似していることがわかった．Yury ら [6] による FSquaDRA フレームワークもリソースを用いて類似アプリを検出した．彼らの分析対象としたアプリ数は $n = 55,779$ だが，その計算量は $O(n^2)$ である．

8.3 本研究と先行研究との主な差異

すでに示したように，我々の *APPraiser* フレームワークはコードベースとリソースベースの両方を組み合わせている．すなわち，リソースベースの手法と効率的なアルゴリズムにより高いスケーラビリティを実現し，コードベースの手法により類似アプリの詳細を明らかにした．アルゴリズムではデータ構造の疎性を利用することで $O(n)$ の計算量を実現し，これは $n = 1,370,000$ のアプリに対しても効率的に動作する．また，こうして得られた類似アプリを関連アプリとクローンの 2 つに分類を行った．このクローンはさらなるサブカテゴリに分けられる．このような詳細な分類により，類似アプリへの対策方法を考えることが出来る．

9 まとめ

本研究の目的は，以下の 2 つの課題を解き明かすことであった．**RQ1**:**いかにしてクローンと関連アプリを区別するか？** **RQ2**: **公式およびサードパーティーマーケットにおけるクローンと関連アプリの実態はどのようなものか？** 我々はまず *APPraiser* フレームワークを用いて類似アプリを抽出し，機械的にクローンと関連アプリを分類することで，**RQ1** を明らかにした．*APPraiser* フレームワークは以下の 3 段階に分けて分析を行う; (1) リソース情報により類似アプリを抽出し， (2) フィンガープリント手法を用いて**関連アプリ**を抽出し， (3) 外部のアンチウイルスソフトとコードの差分分析によって**クローン**の抽出と分類を行う．次に **RQ2** を明らかにするため，公式およびサードパーティーマーケットの 130 万を超えるアプリに対して *APPraiser* フレームワークを適用した．分析を通して得られた主要な知見は以下である; 公式マーケットでは類似アプリの 76%が関連アプリであったのに対し，サードパーティーマーケットでは類似アプリの 45%がクローンであった．また関連アプリの大半は両マーケットにおいて同一開発者によって大量生産されたものであった．また，公式マーケットからサードパーティーにクローンされたアプリのうち，80%が悪性だということも判明した．

本研究の主な貢献は以下のようにまとめられる．我々は Android の “類似アプリ” を**クローン**と**関連アプリ**に二分した．こうした分類によって，内容の重複したアプリに対して適切な対処法を考えることが出来る．また分析対象のデータ量は 130 万を超えるアプリである．これは公式マーケットのもつアプリ数に匹敵する．このような大量のデータを効率的に分析するため，データ構造の疎性を利用した軽量なアルゴリズムを開発した．これにより，$O(n)$ の計算量で類似アプリを抽出することが可能となった．

参考文献

[1] “Stragety analytics.” https://www.strategyanalytics.com/strategy-analytics/blogs/devices/smartphones/smart-phones/2015/03/11/android-shipped-1-billion-smartphones-worldwide-in-2014, Jan. 2015.

[2] AppBrain, “Android operating system statistics.” http://www.appbrain.com/stats/.

[3] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, “Detecting repackaged smartphone applications in third-party android marketplaces,” in *Proc. of the second ACM CODASPY 2012*, pp. 317–326.

[4] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, “Fast, scalable detection of "piggybacked" mobile applications,” in *Proc. of the third ACM CODASPY 2013*, pp. 185–196.

[5] “Fake apps: Feigning legitimacy.” http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-fake-apps.pdf.

[6] Z. Yury, G. Olga, C. Bruno, L. S. Francesco, and M. Ermanno, “Fsquadra: Fast detection of repackaged applications,” Proc. of IFIP DBSec ’14, pp. 131–146, 2014.

[7] N. Viennot, E. Garcia, and J. Nieh, “A measurement study of google play,” Proc. of ACM SIGMETRICS 2014, June 2014.

[8] smali. https://code.google.com/p/smali/.

[9] Y. Yang and J. O. Pedersen, “A comparative study on feature selection in text categorization,” in *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML ’97, pp. 412–420, 1997.

[10] Androguard. https://code.google.com/p/androguard/.

[11] T. Watanabe, M. Akiyama, T. Sakai, H. Washizaki, and T. Mori, “Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps,” in *SOUPS*, 2015.

[12] iBuildApp. http://ibuildapp.com/.

[13] Bizness Apps. https://www.biznessapps.com/.

[14] VirusTotal. https://www.virustotal.com/.

[15] AdAway, “http://adaway.org/hosts.txt.”

[16] Google Play. http://play.google.com/.

[17] Anzhi.com. http://www.anzhi.com/.

[18] J. Oberheide and C. Miller, “Dissecting the android bouncer.” SummerCon, Brooklyn, NY., 2012.

[19] “The gray-zone of malware detection in android os.” https://blog.avast.com/2014/03/31/the-gray-zone-of-malware-detection-in-android-os/.

[20] J. Crussell, C. Gibler, and H. Chen, “Attack of the clones: Detecting cloned applications on android markets,” in *Proc. of the 17th European Symposium on Research in Computer Security*, pp. 37–54, 2012.