博 士 論 文

安全なモバイルアプリの利用および流通を妨げる脅威の分析に関する研究

A Study on Analysis of Threats that Prevent Use and Distribution of Secure Mobile Applications

国立大学法人 横浜国立大学
大学院環境情報学府

金井 文宏
Fumihiro KANEI

責任指導教員 松本 勉 教授

2021 年 3 月

概要

スマートフォンをはじめとしたモバイルデバイスは，現代の人々の生活に根ざした必要不可欠なものになっている．近年のモバイルデバイスでは，従来からのメールや電話などの機能の他にも，特定の用途/機能に特化したモバイルアプリによって様々な機能が提供されており，例えば，ユーザの銀行口座番号のような機微な情報を扱う場面や，制御機器などのクリティカルなシステムの操作に使われる場面など，その用途の多様化が進んでいる．昨今のモバイルアプリの普及は，アプリの流通させる上で不可欠なアプリマーケットや，開発者によるアプリのマネタイズに利用されるオンライン広告など，モバイルアプリを取り巻くエコシステムに支えられている．一方で，これらのエコシステムにおけるセキュリティ上の脅威が問題となっている．

モバイル OS におけるシェアが大きい Android において，攻撃者はリパッケージと呼ばれる手法を用いて，公式マーケットから取得した正規のアプリの中に悪性コードを追加する事で正規のアプリになりすましたマルウェアを作成する．このようなマルウェアは，多くの場合リパッケージの元となった正規のアプリの機能を保持しており，また公式マーケットやサードパーティのマーケット上で正規のアプリになりすまして配布/拡散されるため，ユーザは感染に気づきにくく被害が深刻化する恐れがある．また，攻撃者はリパッケージによるアプリ改ざんの処理を自動化する事で大量のマルウェアを作成し感染を拡大させている．上記のような，モバイルマルウェアによる脅威は，エンドユーザへの直接的な被害だけでなく，アプリの開発者やマネタイズを支えるプラットフォームへの被害も発生させる．アプリの主なマネタイズ手段であるオンライン広告において，広告不正の被害が深刻化している．開発者が継続的にアプリを開発/運用する上でアプリのマネタイズは重要な要素であり，昨今多くのモバイルアプリがオンライン広告を利用したマネタイズを行っている．一方で，アドウェアやボットを用いて広告による収益を詐取する広告不正により本来開発者が得るはずであった広告収益が詐取されるなど，金銭的な被害が発生している．攻撃者がマルウェアをユーザ端末に感染させる目的として，感染端末による不正なマネタイズを行い金銭的な利益を得る事が挙げられる．特に Android においてはリパッケージによるアプリの改ざんが容易であることから，攻撃者は人気の正規アプリ対して不正なマネタイズを行う機能を追加したマルウェアを作成する事で効率的に感染を拡大させ大きな利益を得ていると考えられる．

本研究では，Android のプラットフォームを対象として，攻撃者がマルウェアを用いて不正なマネタイズを行う攻撃への対策を検討する．攻撃者が Android アプリの改ざんにより大量のマルウェア作成しそれらを用いた不正なマネタイズにより利益を得ている背景には，アプリの自動的な改ざんという低コストな攻撃により大きな利益を得ることができるという課題が存在している．そのため本研究では，攻撃者の攻撃にかかるコストを増加させ，かつ攻撃によって得られる利益を減少させる事で，攻撃者による攻撃を非効率なものとして金銭的利益を目的とした攻撃を抑制する対策フレームワークを提案する．

モバイルマルウェアへの対策として，マルウェア検知技術によりアプリマーケット上やユーザ端末においてマルウェアを検知する対策が考えられる．一方で，正規アプリが改ざんにより攻撃に悪用されている現状を踏まえると，アプリマーケットやモバイル端末においてマルウェアを検知する対策と合わせて，正規アプリが攻

撃に悪用されないよう開発段階でアプリの堅牢化を行う対策も合わせて実施する事が望ましい．このようなアプリの堅牢化技術を考える上で，そもそも現状のアプリがどの程度堅牢であり，どのような攻撃に対して脆弱であるのか実態を把握する事が重要である．そこで本研究では，既存の正規アプリがリパッケージに対して，どの程度の耐性を有するかを定量的な検証を行う．検証においては，リパッケージによって作成された実際のマルウェアを解析し，リパッケージ手法を特定した．その後，それらの手法を模して正規アプリに対してリパッケージを行い成功率を評価した．結果として，自動リパッケージの手法により成功率に差がみられるものの，評価対象としたアプリの 7〜9 割においてリパッケージによるアプリ改ざんが成功する事を示す．

次に，アプリマーケット上で Android アプリに改ざん対策を自動的に付与するシステムの構築方式について提案する．提案システムは，マーケット上で流通するアプリに対して一括で自動的に改ざん検知機能を付与する．そのため，アプリ開発者の知識やスキルに依存せずアプリの堅牢化が可能となる．また，提案システムでは，アプリに対してランダム性を有した自己改ざん検知コードを多数挿入する事で，アプリ改ざんへの堅牢性を増加させる．このような対策により，攻撃者によるアプリの自動的な改ざんが困難となり，改ざんによるマルウェアの大量生成を防止する事が可能であると考えられる．

さらに，攻撃者による不正なマネタイズの主な手口である広告不正に着目し，攻撃者の対策回避に対してロバストな新たな特徴量を導入した広告不正検知手法を提案する．提案手法は攻撃者からコントロールが難しい特徴量に基づいて不正な広告リクエストを検知しており，評価の結果，既存の検知技術と比較して攻撃者による対策回避に対してロバストである事を示す．また，提案手法を用いた広告不正の実態調査によって明らかになった，攻撃者がクラウドサービスに管理されている IP アドレスから大量の攻撃を行っている事例や，Windows XP や Android など，特定の OS 環境からの攻撃が多発している事例など，最新の広告不正の特性を述べる．

最後に，アプリ改ざんに起因する不正なマネタイズの脅威への対策フレームワークについて述べる．提案フレームワークは，自動的なアプリ改ざんを困難にする事で攻撃にかかるコストを増加させると同時に，検知回避が困難な広告不正検知手法により攻撃によって得られる利益を減少させる．これにより攻撃者にとってアプリ改ざんによる不正なマネタイズは非効率なものとなり攻撃を辞めざる負えない状況に追い込む事が可能である．アプリ改ざんによって作成したマルウェアによる不正なマネタイズの攻撃が成り立つのは，攻撃者が改ざんにより低コストにマルウェア感染を拡大させる事ができる点や不正なマネタイズの活動が正規のマネタイズの活動と容易に区別できない点を前提にしている．本研究では，提案フレームワークの導入によりこれらの前提を崩す事が可能であり，脅威への対策として有効であることを示す．

Abstract

Mobile devices such as smartphones have played an important role in our lives. On the modern mobile devices, in addition to the traditional functions such as e-mail and phone calls, various functions are provided by mobile applications that are specialized for specific uses/functionality. The uses of mobile apps are becoming more and more diverse, for example, handling sensitive information such as a user's bank account number and operating critical systems such as control equipment.

The popularization of mobile apps is supported by the ecosystem surrounding mobile apps, including app markets, which are essential for app distribution, and online advertising, which is used by developers to monetize their apps. On the other hand, security threats in these ecosystems have become a problem. In the case of Android, which has a large share of the mobile OS market, attackers use a technique called repackaging to create malware that impersonates legitimate apps by adding malicious code to the legitimate apps obtained from the official market. In many cases, such malware retains the functionality of the original app and is distributed from the official market or third-party markets. This makes it difficult for users to notice the infection and potentially causes more serious damage. In addition, attackers create a large amount of malware by automating the process of app tampering through repackaging and spreading the infection. The threats of mobile malware do not only cause direct damage to end users, but also damage to app developers and platforms that support monetization. In online advertising, which is the primary method of monetization for apps, ad fraud has become serious problem. Monetization of apps is an important factor for developers to continuously develop and operate apps, and many recent mobile apps are monetized by using online advertising. On the other hand, ad frauds by adware and bots causes financial damages to app developers by defraud developers of ad revenue. One of the goals of attackers is to gain monetary profits through fraudulent monetization with infected devices. Particularly in the case of Android, since it is easy to repackage apps, attackers are able to efficiently spread their infections and gain significant profits by repackaging popular apps

In this study, I develop countermeasures against attacks which use malware to perform fraudulent monetization in Android platform. In the background of attacks that attackers gain monetary profits from fraudulent monetization by malware created from repackaging android apps, there is an issue that attackers can gain significant monetary profits from low-cost attack such as automated tampering of android apps. Therefore, this paper proposes a framework for countermeasure that discourages attackers from attacking for financial gain by increasing the attacker's cost of attack and decreasing the benefit gained from the attack, thereby making the attacks inefficient.

Detecting mobile malware on device of end-user or on app markets is one of the countermeasure against mobile malware. On the other hand, considering the fact that legitimate apps are compromised by tampering and abused for attacks, in addition to countermeasures by malware detection, it is essential to perform countermeasure which

apply anti-tampering technique to legitimate apps in the development phase. To develop such a anti-tampering technique, it is important to understand how robust the current apps are and what kind of attacks they are vulnerable to in the first place. Therefore, in this study, I quantitatively evaluate the resistance of existing legitimate apps to repackaging. In the evaluation, I analyzed actual malware created by repackaging and identified the methods of repackaging. Then, I performed repackaging to legitimate apps by reproducing these methods and measured the success rate. As a result, I confirmed that I can successfully repackage 70–90% of all tested apps.

Next, I propose a system that automatically adds anti-tampering features to Android apps in the app market. The proposed system automatically adds tamper detection functions to all apps in the market, so it can protect apps without depending on the knowledge and skills of app developers. The proposed system also increases the robustness against app tampering by inserting a number of randomized tamper-detection codes into the app. These measures will make it difficult for attackers to automatically tamper with the app and prevent the mass generation of malware through tampering.

Then, I focus on ad fraud, which is the main method of fraudulent monetization by attackers, and propose an ad fraud detection method that introduces new features that are robust against evasion by attackers. The evaluation of the proposed method shows that the proposed method detects fraudulent ad requests based on features that are difficult to be controlled by attackers, and the proposed method is more robust against evasion by attackers than existing detection techniques. In addition, I describe the characteristics of the latest ad fraud such as a case in which a large amount of fraudulent ad requests was sent from IP addresses assigned to cloud servers, and the client environments extracted from the fraudulent ad requests were heavily biased toward a specific OS version such as Windows XP and Android.

Finally, I describe a framework that prevents threats of fraudulent monetization caused by tampering of Android apps. The proposed framework increases the cost of attacks by making automated tampering of Android apps more difficult, while reducing the benefits of attacks by performing ad fraud detection that is difficult to avoid. This makes attacker's fraudulent monetization of app tampering inefficient and forces them to discontinue their attacks. To efficiently earn the monetary benefits from attacks by fraudulent monetization from malware created from tampering of apps, attacker must be able to spread malware infections at low-cost and malicious monetization activities cannot be easily distinguished from legitimate monetization activities. In this paper, I show that the proposed framework can break these assumptions and is efficient against these threats.

目次

図目次

表目次

第 1 章

序論

1.1 モバイルアプリの利用・流通を支えるエコシステム

スマートフォンをはじめとしたモバイルデバイスは，現代の人々の生活に根ざした必要不可欠なものになっている．近年のモバイルデバイスでは，従来からのメールや電話などの機能の他にも，特定の用途/機能に特化したモバイル端末向けアプリケーション（以降，モバイルアプリ）によって様々な機能が提供されている．モバイルアプリの市場は年々増加傾向にあり，総務省の報告 [64] によれば，2019 年には 560 億件のアプリダウンロードが行われ，売上は 458 億ユーロに達したとされている．加えて，モバイルアプリの用途も日々多様化が進んでおり，昨今ではユーザの銀行口座番号のような機微な情報を扱う場面や，制御機器などのシステムの操作に使われる場面など，クリティカルな用途でモバイルアプリが利用されるケースも増えてきている．

図 1.1 に，開発者によって作成されたモバイルアプリがエンドユーザに利用される流れと，ユーザがアプリを利用する中で発生する収益の流れを示す．モバイルアプリは，一般的に開発者にアプリマーケットを通じてエンドユーザのデバイスにインストールされる．また，開発者はアプリのマネタイズを行うことで，エンドユーザによるアプリ利用を通して収益を得る．このようなアプリの流通，およびアプリの利用により収益が発生するサイクルは，世の中でモバイルアプリが継続的に利用されていく上で不可欠であり，これらのサイクルはアプリマーケットやオンライン広告など，モバイルアプリを取り巻くエコシステムによって実現されている．

1.1.1 アプリの流通とマーケット

昨今のモバイルアプリが世の中に流通する上で，アプリマーケットは必要不可欠な存在である．アプリマーケットとは，モバイルデバイスの OS/フレームワークなどを開発しているベンダーや，サードパーティによって運営されている，アプリを配信する為のプラットフォームである．

Android においては Google によって運営されている GooglePlay [25]，iOS においては Apple によって運営されている AppStore [10] が，それぞれの OS の公式アプリマーケットである．また，上記のような公式アプリマーケット以外にも，サードパーティによって運営されているアプリマーケットも存在する．例えば，中国においては Android の公式アプリマーケットである GooglePlay へのアクセスが制限されている事から，多くのサードパーティアプリマーケットが存在している．なお，iOS においては，開発者がアプリの開発時にテスト目的で開発中のアプリをインストールするケースを除いて，原則として公式アプリマーケットである AppStore 以外からのアプリのインストールを禁止している．また，Android においては，ユーザが明示的に

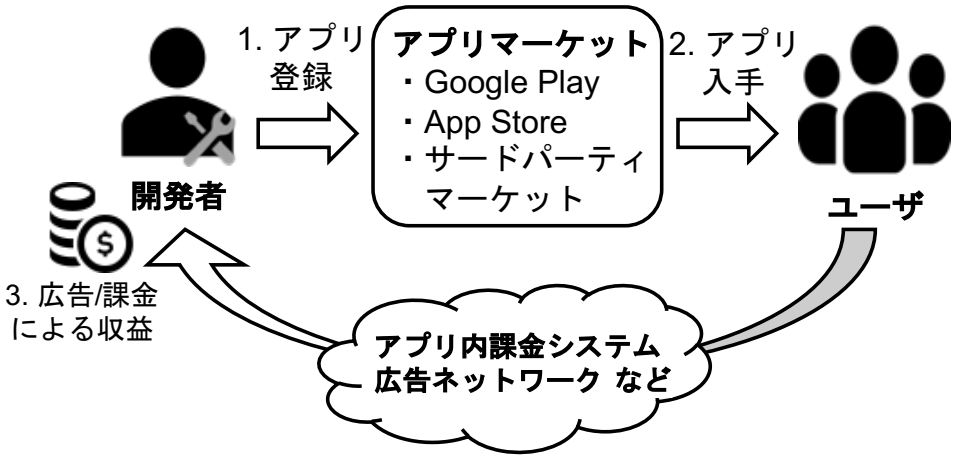

図 1.1　アプリの流通と収益の流れ

公式マーケット以外からのアプリのインストールを許可する設定変更を行わない限り，公式マーケットである GooglePlay 以外からのアプリのインストールができないようになっている．

開発者がマーケット上で自身のアプリを公開/配信したい場合には，マーケットに対してアプリの登録を行う．開発者がアプリの登録申請を行ってから実際にアプリがマーケット上で公開されるまでの流れは様々であるが，例えば，アプリマーケット側がアプリの登録申請に際して開発者の個人情報の提出を求める場合や，公開予定のアプリやアプリのソースコードに対してアプリマーケットが手動/自動での検査を実施する場合など，攻撃者によって不正なアプリが登録されてしまう事を防止する為の取り組みがアプリマーケットによって行われている場合がある．

1.1.2　アプリマネタイズとオンライン広告

開発者が継続的にアプリの開発/運用を続けていく上で，アプリのマネタイズは重要な要素の一つである．開発者がモバイルアプリを公開し流通させる中で収益を得る手段は，有料アプリ，アプリ内課金，アプリ内広告に大別される．有料アプリは，アプリ自体を有料で公開する方法である．この場合ユーザは料金を支払うことでアプリのインストールが可能となる．アプリ内課金では，ユーザはアプリを利用する中で例えば特定の機能を有効にしたり，アプリにおける追加コンテンツを入手する為に料金を支払う．アプリ内広告では，ユーザがアプリの利用中に広告が表示され，これらの広告による広告収益が最終的に開発者に対して支払われる．また，開発者は上記のうち複数の手段を組み合わせてアプリのマネタイズを行う場合もある．

昨今のモバイルアプリにおいては，アプリ内課金とアプリ内広告によるマネタイズが広く採用されている．App Annie 社の報告 [9] によれば，2019 年ではアプリ内課金によって 120 億ドル，アプリ内広告によって 190 億ドルの収益が発生したとされている．このように，モバイルアプリには課金や広告による大規模な市場が存在しており，それらを支えるアプリ内課金のシステムや広告配信を行うプラットフォームは必要不可欠なものであると言える．

1.2 モバイルアプリにおける脅威

モバイルアプリの急速な普及に伴い，それらの利用/流通を支えるエコシステムにおける様々な脅威が問題となっている．特に，主要なモバイル OS の一つである Android は，OS の仕様がオープンである事や多くのサードパーティマーケットが存在することから攻撃の標的となりやすく被害が拡大している．

アプリを利用するエンドユーザを標的とした脅威としてモバイルマルウェアが挙げられる．モバイルマルウェアは，モバイル端末上で動作し悪意のある動作を行うソフトウェアの総称である．セキュリティベンダである Kaspersky 社の報告 [36] によれば，2018 年には 500 万種のモバイルマルウェアが検知され，990 万人のユーザがモバイルマルウェアによる攻撃に遭遇したとされている．モバイルマルウェアは，多くの場合モバイルアプリとしてユーザの端末に感染し悪性な活動を行う．出現当初は比較的単純な構造をしていたモバイルマルウェアであるが，最近では通信の暗号化や，コードの難読化，解析環境の検知など，検知・解析を逃れるために，より高度な技術が使用されるようになってきている事が報告されている [60]. また攻撃者は，既にパッケージ化された Android アプリに対して新たにコードを挿入する，リパッケージと呼ばれる手法を利用してマルウェアを作成する場合がある．これらはリパッケージマルウェアと呼ばれる．パッケージマルウェアは，悪性コード挿入前の元の正規アプリの機能も保持した上で悪性な活動を行うため，ユーザはマルウェア感染に気づきにくく被害が深刻化する恐れがある．既存研究 [60] では，8 割以上の Android マルウェアがリパッケージによるアプリ改ざんによって作成されている事が報告されている．また，Symantec による報告 [13] によれば，中国のサードパーティ製アプリマーケットで確認された“Android.Troj.mdk”というファミリー名のリパッケージマルウェアにおいては，7000 種類以上の正規アプリに対してリパッケージが行われ，最大 100 万台のデバイスに感染したとされている．上記の事例のように，攻撃者が大量のリパッケージマルウェアを作成する際には，その処理を自動化することで被害を拡大させていると考えられる．

上記のような，モバイルマルウェアによる脅威は，エンドユーザへの直接的な被害だけでなく，アプリの開発者やマネタイズを支えるプラットフォームへの被害も発生させる，アプリの主要なマネタイズ手段であるオンライン広告において広告不正による被害が深刻化している．広告不正とは，実際には広告効果が発生しない不正な手段を用いて広告収益を詐取する攻撃の総称である．広告不正によって，本来アプリの開発者が受け取るはずであった広告収益が攻撃者によって詐取される場合や，広告不正が蔓延することにより広告配信を行うプラットフォームベンダーの信頼が失われ結果として経済的な被害が発生する場合など，直接/間接的な被害が発生する．Interactive Advertising Bureau （IAB）の報告 [8] によれば，2018 年に米国で発生した広告不正によって 8.2 億ドルの被害が発生したとされる．広告不正は，たとえばアドウェアや PUP(Potentially Unwanted Program)，ボットに感染したユーザの端末において，画面上に大量の広告を表示させたり，ユーザの操作を介さずに広告クリックを行なうことで行われる．また，攻撃者は広告不正による攻撃を検知されにくくする為に，広告不正による通信を正規ユーザによる通信に見せかけて攻撃を行う．既存研究 [43] では，攻撃者が広告不正を行う際の通信を分散させることで，検知の回避を試みている事が指摘されている．

攻撃者がマルウェアをユーザ端末に感染させる目的として，感染端末による不正なマネタイズを行い金銭的な利益を得る事が挙げられる．また，前記の通り Android においてはリパッケージによるアプリの改ざんが容易であることから，正規アプリ対して不正なマネタイズを行う機能を追加したマルウェアによる被害が発生している．FireEye 社による調査によれば，Android を標的としたマルウェア “kemoge” はアプリ改ざんによる不正な広告モジュールの埋め込みによって作成され，20 カ国以上で被害が確認された事が報告されている [23]．図 1.2 に，攻撃者が Android マルウェアを利用して不正なマネタイズを行う際の流れを示す．攻撃者

はアプリマーケットから人気のある正規アプリを入手した上で，それらに対して改ざんを行い不正なマネタイズを行うためのマルウェアを作成しユーザに感染させる．この際，攻撃者は改ざんを自動化することで大量のマルウェアを作成する事でマルウェアを感染/拡散を効率的に行っていると考えられる．ユーザ端末に感染したマルウェアは，オンライン広告などのマネタイズインフラに対して不正なリクエストを送信する事で収益を詐取する．

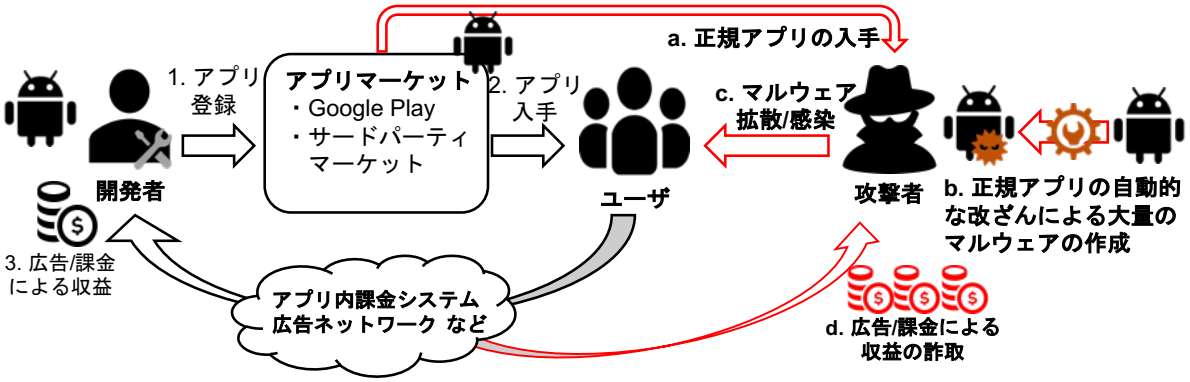

図 1.2　Android マルウェアによる不正なマネタイズにより攻撃者が金銭的な利益を得る流れ

1.3　研究の目的

本研究では，マルウェアによる被害が深刻である Android のプラットフォームを対象として，攻撃者が Android マルウェアを用いて不正なマネタイズを行う攻撃への対策を検討する．

Android マルウェア対策への対策として，マルウェア検知技術によりアプリマーケット上やユーザ端末においてマルウェアを検知する対策が考えられる．一方で，正規アプリが改ざんにより攻撃に悪用されている現状を踏まえると，アプリマーケットやモバイル端末においてマルウェアを検知する対策と合わせて，正規アプリが攻撃に悪用されないよう開発段階でアプリの堅牢化を行う対策も合わせて実施する事が望ましい．このようなアプリの堅牢化技術を考える上で，そもそも現状のアプリがどの程度堅牢であり，どのような攻撃に対して脆弱であるのか実態を把握する必要がある．本研究では，攻撃者がリパッケージにより正規アプリを改ざんし攻撃に悪用している状況を踏まえ，現状の正規アプリがリパッケージに対してどの程度耐性を有しているか定量的な評価を実施する．

リパッケージによるアプリ改ざんへの対策として開発段階でのアプリの堅牢化が有効であると考えられる一方で，このような対策は，開発者が自発的に対策付与を行う必要があり，実装の際にはアプリ改ざんに関する知識や実装能力が要求される．このため，個々の開発者のセキュリティに対する意識や実装スキルに依存して，そもそも対策が付与されない場合や，対策の強度が弱く容易に回避されてしまう場合がある．本研究では，個々のアプリ開発者に依存せず堅牢なアプリ改ざん対策を自動的に付与する対策の具体的な実現方式を提案する．また本方式によるアプリの堅牢化によって，正規アプリの改ざんにかかるコストを増加させ，攻撃者による自動的なアプリ改ざんを用いた大量のモバイルマルウェア作成を困難にさせる事が可能である事を示す．

攻撃者がマルウェアにより不正なマネタイズを行う際には，Android における主要なマネタイズ手段の一つであるオンライン広告を標的として広告不正による攻撃を行っていると考えられる．広告不正への対策として，利益の詐取につながる攻撃者からの不正なリクエストを検知する対策が考えられる．このような対策を考

える上で，マネタイズのプラットフォームを提供するベンダーの視点から攻撃を観測し網羅的な検知を行う事が効果的である．またこのような対策に対して，攻撃者は不正なリクエストを正規のリクエストに見せかけることで対策の回避を試みるため，対策の回避に対して堅牢な検知手法が望ましい．本研究では，広告配信プラットフォームの視点から不正なリクエストを検知する手法を提案する．また，攻撃者がコントロールする事が困難な特徴量を用いて検知を行うことで，提案手法が攻撃者による対策回避に対して堅牢である事を示す．

攻撃者がモバイルアプリの改ざんにより大量のマルウェア作成しそれらを用いた不正なマネタイズにより利益を得ている背景には，アプリの自動的な改ざんという低コストな攻撃により大きな利益を得ることができるという課題が存在している．本研究では，自動的なアプリの改ざんを防止する対策技術とオンライン広告における対策回避が困難な広告不正検知技術の組み合わせにより，アプリ改ざんに起因する不正なマネタイズへの対策フレームワークを構築できる事を示すと共に，提案フレームワークの有効性の根拠となる攻撃の前提や，提案フレームワークの社会実装に向けた課題について述べる．

1.4 研究の貢献

Android アプリの自動リパッケージに対する耐性評価（ 3 章）

はじめに，Android アプリへの攻撃手法であるアプリのリパッケージに着目した分析を実施した．攻撃者がリパッケージによりマルウェアを大量に作成する際には，リパッケージ処理の自動化が必須であると考えられるが，自動リパッケージの実態や対策については，十分な調査・検討が行われていない．そこで，既存の正規アプリが自動リパッケージに対して，どの程度の耐性を有するか検証を行った．具体的には，実際のリパッケージによって作成されたマルウェア（以降，リパッケージマルウェア）を解析し，リパッケージ手法を特定した上で，それらの手法を模して複数の正規アプリに対して，外部と通信を行う機能を持つ検証用コードを挿入し攻撃者によるリパッケージを再現した．その後，作成したリパッケージ済みアプリを動的解析して，挿入した検証用コードが正常に動作するかどうかを検証した．結果，自動リパッケージの手法により成功率に差がみられるものの，評価対象としたアプリの 7〜9 割において，挿入した検証用コードが正常に動作し，尚且つ起動時の動作が変化しない事が明らかになった．この実験において挿入した検証用コードを，悪性のコードに変更した場合でも，同様の方法で自動リパッケージが可能であることが予想される．以上より，現状の Android アプリの多くは自動リパッケージへの耐性が不十分であり，耐タンパー技術等を用いたリパッケージ対策が必要であることを示した．

Android アプリの改ざん対策技術の方式検討（ 4 章）

次に，Android アプリに対して自動的に改ざん対策を付与する技術の方式検討を行った．リパッケージによるアプリ改ざんへの対策として開発段階でのアプリの堅牢化が有効であると考えられる．一方で，このような対策は開発者が自発的に対策付与を行う必要があり，開発者の知識や実装スキルに依存して対策が実施されない場合がある．これらを踏まえ，アプリマーケット上で一括で自動的に改ざん対策を付与するシステムの構成方法について提案した．提案方式は既存自動的な改ざん対策付与技術を拡張する事で実現可能であり，アプリに自己改ざん検知機能を自動的かつ攻撃者によって回避されにくいように挿入可能である．

広告不正の検知手法の提案および実態調査（ 5 章）

さらに，アプリのマネタイズに利用されるオンライン広告を標的とした攻撃について分析を実施した．モバイルアプリや Web サイトの収入源であるオンライン広告を悪用した広告不正に対して，様々な対策技術が研究されている．一方で既存研究では，広告によって生じる通信や，広告が表示される Web サイトやモバイルアプリを対象とした分析が行われているが，広告を閲覧するクライアントの立場からの分析では，実際に発生

している広告不正の手口や傾向を調査する事が困難である．加えて，既存研究における広告不正検知技術は，攻撃者によってコントロールが容易な特徴（例：広告リクエストのバースト性）を検知に用いており，回避が容易である問題がある．これらを踏まえ，広告ネットワークに到達するユーザからの広告リクエストの観測に基づいて，攻撃者の対策回避に対してロバストな新たな特徴量を導入した広告不正検知手法を提案した．評価の結果，提案手法はリクエストのバースト性に基づいた既存の検知手法より多くの広告不正を検知可能である事を確認した．また，提案手法を実際の大規模な広告リクエストログに対して適用することで，昨今の広告不正の特徴を明らかにした．

アプリ改ざんに起因する不正なマネタイズへの対策フレームワークの提案（6 章）

最後に，アプリ改ざんに起因する不正なマネタイズへの対策フレームワークを提案した．提案する脅威対策フレームワークでは， 3 章， 4 章での発見/提案を基にマーケット上で流通する正規アプリに対して改ざん対策を自動的に付与し，攻撃者が改ざんにより大量のモバイルマルウェアを低コストに作成する事を困難にさせる．また，同様に 5 章での提案を基に，攻撃者が攻撃によって得られる金銭的な利益を減少させる．これらの対策の組み合わせにより，攻撃者に対して攻撃コストの増加および攻撃によって得られる利益の減少を強いる事で，攻撃者にとって攻撃が非効率なものとなり，結果として金銭的利益を目的とした攻撃を抑制する事が可能である事を示した．また，提案する脅威対策フレームワークを実社会に実装する上で，今後検討が必要な課題やアプリの流通/マネタイズに関わるステークホルダーとの連携の必要性について論じた．

1.5 本論文の構成

本論文の構成を図 1.3 に示す．まず，2 章では本研究に関する関連研究について述べる．3 章ではリパッケージを用いたアプリへの攻撃に着目した脅威の観測・分析として実施した，Android アプリの自動リパッケージに対する耐性評価の結果について述べる．4 章では，3 章で得られた結論を元に，Android アプリに対して自動的に堅牢な改ざん対策を付与するシステムの構築方法について論じる．5 章では Android アプリの主要なマネタイズ手段であるオンライン広告に着目し，対策回避に対して堅牢な広告不正の検知手法を提案する．また提案手法を用いて実施した広告不正に関する実態調査の結果について述べる．6 章では，アプリ改ざんに起因する不正なマネタイズへの対策フレームワークについて述べる．最後に，7 章で本研究の結論について述べる．

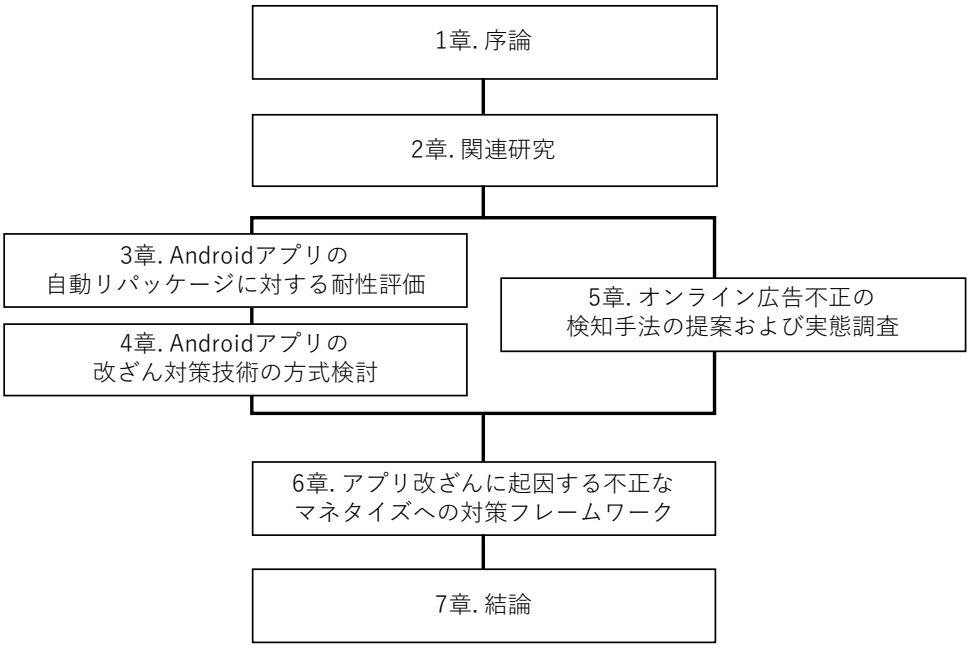
1章. 序論
2章. 関連研究
3章. Androidアプリの
自動リパッケージに対する耐性評価
4章. Androidアプリの
改ざん対策技術の方式検討
5章. オンライン広告不正の
検知手法の提案および実態調査
6章. アプリ改ざんに起因する不正な
マネタイズへの対策フレームワーク
7章. 結論

図 1.3　本論文の構成

第 2 章

関連研究

2.1 マルウェア検知/分析技術

モバイルマルウェアへの対策の研究として，動的/静的解析によるマルウェアの検知技術が多数提案されている．

アプリの静的解析技術に関して，Arzt らの論文 [11] では，静的解析によりアプリの実行コードにおけるデータフローを解析し，アプリによる情報漏えいを検知するシステム，FlowDroid が提案されている．また Li らの論文 [38] では，FlowDroid を拡張することで，複数のアプリが共謀して情報漏えいを行うような攻撃の検知システム，ICCTA を提案している．同様に，機械学習によるアプローチとして，Mariconti らの論文 [41] は，アプリの実行コードにおける API 呼び出しの順序情報を特徴量として学習を行うことでマルウェアを検知するシステム，MAMADroid を提案している．

アプリの動的解析技術に関して，Enck らの論文 [22] は，アプリの実行時に機微なデータに対してタグ付を行いそれらの伝搬を追跡するテイント解析により，情報漏えいを検知する動的解析システム，TaintDroid が提案している．また，Wong らの論文 [54] は，アプリの動的解析においてユーザ操作に起因する入力やイベントを模擬することで，解析時に実行されるコードのカバレッジを向上させるシステム，IntelliDroid を提案している．

2.2 Android アプリのリパッケージ

攻撃者によるアプリのリパッケージに着目した分析も多数存在する．Zhou らの論文 [59] は，Fuzzy Hashing の技術を用いた DroidMOSS と呼ばれるシステムを構築し，公式マーケットから収集した正規アプリのセットを，サードパーティマーケット内のアプリと比較することで，サードパーティマーケット内のリパッケージされたアプリを検出する手法を提案している．また，サードパーティマーケットから収集したアプリの内，5%〜13% のものがリパッケージによって作成されたとしており，リパッケージの主な内容としては，広告モジュールの追加・改変や悪意のあるペイロードの追加が見られた事を報告している．また Nicolas らの論文 [53] は，GooglePlay 上のアプリを高効率に収集可能なマーケットクローラ PlayDrone を提案し，大規模なアプリ収集を行った上で各アプリに含まれるリソースの類似度からリパッケージによって作成されたと考えられるアプリを検出している．

2.3 アプリの改ざん対策技術

モバイルアプリ自身に改ざん対策を付与する事でアプリの堅牢化を図った既存研究も行われている．Protesenko らの論文 [48] では，ネイティブコードを利用した実行コードの難読化，および自己改ざん検知手法が提案されている．この手法では，Android におけるバイトコードである Dalvik バイトコードをクラス単位で複数のセグメントに分割した上で予め暗号化しておき，アプリの実行時には，ネイティブコード上に実装された復号ルーチンから動的に Dalvik バイトコードの復号および完全性の検証を行う．これにより，アプリ本来の機能が実装された Dalvik バイトコードはアプリの実行時にしか復号されず，静的なアプリ改ざんに対して堅牢となる．Divilar [58] はオペコードのランダム化，及び仮想インタプリタの自動生成による難読化手法である．この手法では，実行コードのオペコードをランダムな値に変換しておき，実行時には変換ルールに基づいて自動生成された仮想的なインタプリタがオペコードを逆変換しながら読み込む．これにより，既存ツールを用いた静的／動的解析が無効化される．また，Luo らの論文 [40] は，自己改ざん検知機能をアプリに対して自動的に付与する手法である SSN を提案している．SSN では，アプリのソースコードに対して改ざん検知用のコードがランダム性に基づいて多数挿入される．また，非決定性プログラミングのコンセプトを用いており，改ざんが検知された際に確率的な対応を行うことで，動的解析によるアプリのデバッグを困難にしている．

2.4 アプリマーケットにおける脅威の観測/分析

アプリマーケットにおけるセキュリティ対策の実態や，マルウェアの拡散状況の調査など，アプリマーケットに着目した観測/分析技術の研究も盛んに行われている．Zhou らの論文 [61] は，アプリが要求するパーミッションの分析によりアプリマーケット上からマルウェアを検知するシステム，DroidRanger を提案し，GooglePlay およびサードパーティマーケット上から未知マルウェアを含む複数のマルウェアファミリーを検出している．Kikuchi らの論文 [37] は，GooglePlay と 4 種類のサードパーティアプリマーケットから収集したアプリに対してアンチウイルススキャンを用いた検査を行うことで，各マーケットにおけるマルウェアの含有率やマルウェアの生存期間について比較分析を行っている．また分析の結果，特定のサードパーティアプリマーケットにおいてマルウェアが長期間削除されずに掲載され続けており，攻撃者がセキュリティ対策が不十分なマーケットを利用してマルウェアを拡散させている事が考察されている．Ishii らの論文 [33] では，サードパーティアプリマーケットにおけるセキュリティ対策の実施状況を調査する為に，世界各国における 27 種類のマーケットからアプリを収集した上で，それらの悪性判定や複数マーケットにおけるアプリのマルチリリースの実態を明らかにしている．

2.5 アプリのマネタイズを支えるプラットフォームにおける脅威の観測/分析

開発者がマネタイズを行う上で利用されるプラットフォームである，アプリ内課金やオンライン広告に着目した脅威の観測/分析技術が研究されている．Mulliner らの論文 [45] では，Android SDK に付属しているアプリ内課金を実装するためのライブラリの分析により，アプリ内課金の処理を自動的にバイパス可能な攻撃が存在する事を明らかにし対策手法を提案している．また，Yang らの論文 [56] は，サードパーティ製のアプリ内

課金実装用ライブラリを分析し，安全なアプリ内課金の処理のモデル化を行うと共に，実態調査を通じて実在のアプリにおいて課金処理をバイパス可能な脆弱性が存在する事を示した．加えて，アプリ内課金実装用ライブラリにおけるセキュリティ上の欠陥や，それらを利用するためのドキュメントにおける不備についても明らかにした．

アプリが利用するオンライン広告におけるセキュリティ研究は，広告を利用した不正な活動の実態調査やそれら脅威の検知を目的としている．Crussell らの論文 [16] は，アプリの動的解析により発生する広告通信を分析し，不正な広告リクエストを検知する手法を提案している．また，提案手法を用いて実在のアプリを対象とした大規模な分析を行うことで，バックグラウンドでの広告表示や，自動的な広告のクリックなど，モバイルアプリを標的とした広告不正の実態を明らかにしている．また，上記を含めた複数の既存研究 [19, 42, 50, 57] において，広告によって発生する通信における異常性を特徴量として，広告不正を検知する手法が提案されている．これらの既存研究において広告不正を検知する為に利用される特徴としては，通信の頻度や順序情報，発生する広告収益のバースト性などが挙げられる．また，これらの広告不正検知を目的とした研究の多くは，広告が表示されるアプリや Web ブラウザなどクライアント側での分析が行われている．

2.6 関連研究と本研究の差異

関連研究と本研究の違いについて述べる．

モバイルマルウェアへの対策や実態調査に関する研究では， 2.1 節， 2.4 節に示す通り，マルウェアの検知/解析技術の提案やアプリマーケットにおけるセキュリティ対策の状況およびマルウェアの拡散の実態調査などに焦点が当てられている．本研究では正規アプリが攻撃に悪用されないよう開発段階でアプリの堅牢化を行う対策に着目しており，これらの研究とは目的が異なる．また，これらの研究で提案/評価が行われているモバイルマルウェア対策と本研究で着目するアプリの堅牢化対策の組み合わせにより，より安全なアプリの利用および流通に貢献できる．

正規アプリの改ざんによる脅威に着目した研究としては， 2.2 節に示す通り，正規アプリとリパッケージによって改ざんされたアプリの比較によってマーケット上で不正なアプリの検知する事を目的としたものが主である．しかし，これらの研究では，そもそも現状のアプリがどの程度の堅牢性を有しているかが十分に評価されておらず，中でも攻撃者によるアプリ改ざんに対して，正規アプリがどの程度耐性を有しているかは明らかになっていない．

アプリのマネタイズを支えるプラットフォームにおける脅威の観測/分析については，2.5 節に示す通り，アプリやライブラリの分析に基づいたアプリ内課金システムへの攻撃検知や，モバイルアプリ/Web ブラウザなどクライアント側のアプリケーションの分析に基づいたオンライン広告不正の検知/観測が行われている．これらの研究ではアプリ内課金や広告表示を行うクライアント側での分析に焦点が当てられているため，分析可能な攻撃が限られており対策の網羅性に課題がある．加えて，オンライン広告不正検知に関する既存研究では，攻撃者の立場からコントロール可能な特徴に基づいた検知が行われており，対策回避に対して脆弱である課題がある．

本論文ではこれらを踏まえ以下の研究を実施した．まず，正規アプリの改ざんへの耐性が不明瞭である課題を踏まえ，第 3 章にて，Android アプリの自動リパッケージに対する耐性評価を実施した．これらの結果から，現状のモバイルアプリがリパッケージによる改ざんに対して耐性を有していない事を示すと共に，被害の防止に向けて必要な対策について考察した．加えて，第 4 章にて，アプリ開発者の知識やスキルに依存せず，全自動でアプリに対して改ざん対策を付与するシステムを提案した．

また，アプリのマネタイズプラットフォームにいて攻撃観測の網羅性の課題や，既存の対策が攻撃者による対策回避に対して脆弱である課題を踏まえ，第 5 章にて，オンライン広告不正の検知手法の提案，および検知手法を用いた広告不正の実態調査を行った．提案手法では攻撃者からコントロールが難しい特徴に基づいて広告不正の検知を行っており，評価の結果既存手法と比較して攻撃者による対策回避に対して堅牢である事を示した．また，提案手法を実際の広告配信プラットフォーム上で観測されたデータに対して適用することで，昨今の広告不正における攻撃の特徴を明らかにした．

第 3 章

Android アプリの自動リパッケージに対する耐性評価

3.1 はじめに

近年，スマートフォン向け OS として Android が広く用いられている一方で，Android を狙ったマルウェアの数も増加している．Android マルウェアには，既にパッケージ化されたアプリに対して新たにコードを挿入する，リパッケージと呼ばれる手法を利用し，正規のアプリに対して悪性コードを挿入することで作成されたものがある．これらはリパッケージマルウェアと呼ばれる．リパッケージマルウェアは，悪性コード挿入前の元の正規アプリの機能も保持した上で，多くの場合，サードパーティマーケット上で拡散される．攻撃者は公式マーケットである GooglePlay 上の人気の正規アプリに対してリパッケージを行い，アプリ審査等のセキュリティ機構が不十分なサードパーティマーケット上にアップロードすることで，被害を拡大させていると考えられる．リパッケージマルウェアの機能は，Bot 化，RAT 化のようなものに加え，典型的な情報摂取機能も備わっている場合も多く，感染ユーザのプライバシへの被害が懸念される．リパッケージマルウェアの被害事例としては，“Android.Troj.mdk”というファミリー名のマルウェアによる事例 [13] が挙げられる．このマルウェアは，中国のサードパーティ製マーケットにて配布されたリパッケージマルウェアであり，7000 種類以上の正規アプリに対してリパッケージが行われ，最大 100 万台のデバイスに感染したとされている．

上記の事例のように，攻撃者が大量のリパッケージマルウェアを作成する際には，その処理を自動化している事が予想されるが，自動リパッケージの実態やその対策については，十分な検討が行われていない． 本章では，既存の正規アプリが自動リパッケージに対してどの程度の耐性を有するかを検証するための実験を行った結果を示す．まず，実際のリパッケージマルウェアの解析を行うことでリパッケージの方法を特定し，自動リパッケージを再現するスクリプトを作成した．次に，Web 上から収集した複数の正規アプリに対して，外部と通信を行う機能だけを持つ検証用のコードを挿入することで，簡易的に自動リパッケージを行った．さらに，作成したリパッケージ済みアプリを動的解析することで，挿入した検証用コードが正常に動作するかどうかを検証した．その結果，自動リパッケージの際に用いた手法によって成功率に差が見られたものの，元にした正規アプリの内の約 75%〜90% において，挿入した検証用コードが正常に動作し，尚且つ元の正規アプリの起動時の動作に変化が生じないことを確認した．さらに，インストール数の観点から，リパッケージが行われる事によって大きな被害をもたらすと考えられる 33 種類のアプリにおいて，アプリの持つ基本的な機能がリパッケージ後にも保持されるかを確認し，87.9 %にあたる 29 種類のアプリにおいて機能が保持されている事を確認した．本実験においては，単純な機能のみを持つ，非悪性の検証用コードを挿入したが，これを実際

に攻撃者が利用するような悪性のものに変更した場合でも，同様の方法で自動リパッケージが可能であることが予想される．以上より，現状の Android アプリの多くは自動リパッケージへの耐性が不十分であり，耐タンパー技術等を用いたリパッケージ対策が必要であることを示す．

3.2 Android アプリの構造とリパッケージ

本節では，Android アプリの構造，及びリバースエンジニアリングの手法について説明し，攻撃者によるリパッケージの概要について述べる．

3.2.1 Android アプリの構造

Android アプリは，apk ファイルという zip 形式で圧縮された専用ファイルとして配布されている．apk ファイルの主な中身としては，アプリのメタ情報をまとめた AndroidManifest.xml ファイル，Android 上で実行されるバイトコードの本体である classes.dex ファイル，画像や音声等のリソースファイル，署名情報が保存されている META-INF フォルダ等が挙げられる．それ以外にも，ネイティブライブラリを含んだ lib フォルダや，アプリ作成者が任意のファイルを置くことができる assets フォルダなどが存在する場合もある．

Android アプリは，基本的に Java　によって開発が行われ，Java のソースコードが Java バイトコードにコンパイルされた後に，Android 上で動作する Dalvik VM 用のバイトコードである dex 形式のファイルへと変換が行われる．AndroidManifest.xml 内に記述される情報としては，端末内でアプリを一意に定める ID であるパッケージ名，利用するパーミッション，アプリケーションに含まれているアクティビティ，サービス，コンテントプロバイダ等のコンポーネント，それぞれのコンポーネントが実装されているクラス名などが挙げられる．また，Android におけるアプリ間連携機能であるインテント [24] に関して，それぞれのコンポーネントが受信するインテントの種類が，インテントフィルタとして AndroidManifest.xml ファイル内に記述される．

3.2.2 Android アプリのリバースエンジニアリング

前述の通り，Android アプリの多くは Java によって記述されており，他の言語と比較して，リバースエンジニアリングが容易であるという特徴がある．具体的には，apktool [32] や smali/baksmali [34] 等のツールを用いることで，バイトコードである classes.dex ファイルを可読であるテキスト形式のファイルにディスアセンブルすることが可能である．図 3.1 は apktool のディスアセンブルによって出力されるファイルの構成の一例を示しており，/smali フォルダ以下に含まれている.smali ファイルがテキスト形式にディスアセンブルされたコードに当たる．apktool によって出力された各ファイルに対して，コードの改変やリソースファイルの追加を行った後に，再び apktool を用いて apk ファイルへとビルドすることも可能である．再ビルド後には jarsigner [46] のコマンドを用いて apk ファイルに対して署名付与を行うことで，再び Android 端末へのインストールが可能となる．

3.2.3 Android アプリのリパッケージ

攻撃者がリパッケージを行う際には，前項で述べたようなリバースエンジニアリングの技術を用いて，正規アプリをディスアセンブルした後に，悪性コードの挿入，再ビルドを行っていると予想される．

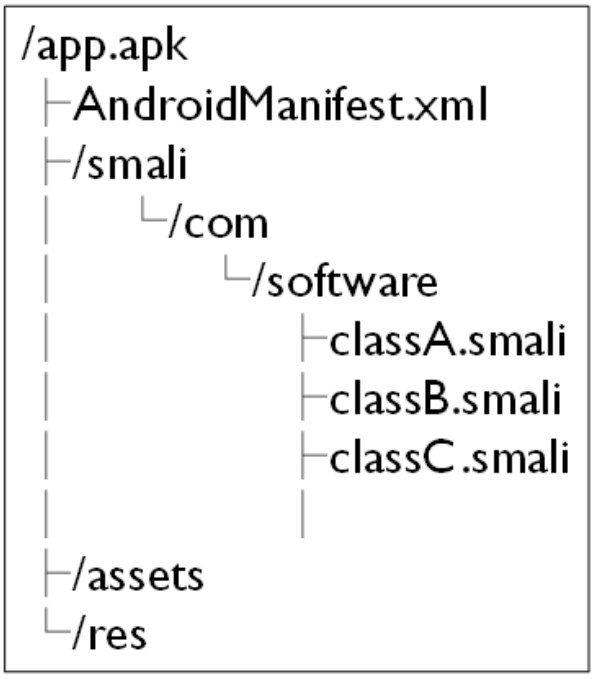

図 3.1　apktool のディスアセンブルによって出力されるファイルの構成例

悪性コードの挿入に関しては，挿入されるコードの持つ機能によって様々なパターンが考えられる．例えば，攻撃者が悪性コードを含んだ Java パッケージ（以下，悪性パッケージ）を作成し，正規アプリに挿入する場合を考えると，挿入される悪性パッケージ内で，元の正規アプリでは利用されていなかったパーミッションやコンポーネントが利用されていた場合，AndroidManifest.xml 内に，それらを定義する記述を追加する必要がある．また，悪性コードのエントリポイントとして，元の正規アプリのコード内に，挿入された悪性コードを呼び出す処理を追加する場合なども考えられる．

上記の例のように，攻撃者によって悪性コードが挿入される際には，実際に悪性な処理を行うコードやパッケージを挿入するのに加えて，AndroidManifest.xml の改変や，元の正規アプリに含まれていたコードの改変を行う事が予想される．

3.2.4　リパッケージの自動化

自動的なリパッケージの場合を考えると，個々のアプリの内部構造に応じたコードの挿入手法を自動化するのはコストが高いため，機械的に解析が可能な情報（AndroidManifest に含まれる情報や，アプリの Java パッケージの構造など）を利用して多くのアプリに対して共通の処理でリパッケージが可能な手法が用いられると考えられる．これらの制約を考慮すると，挿入される悪性コードの呼び出され方や正規アプリ内で改ざんされる箇所などに傾向が現れる事が予想される．

表 3.1　評価対象の正規アプリの利用するパーミッション（上位 10 位まで）

パーミッション名	アプリ数
android.permission.INTERNET	1545
android.permission.ACCESS_NETWORK_STATE	1445
android.permission.WRITE_EXTERNAL_STORAGE	1087
android.permission.WAKE_LOCK	898
com.google.android.c2dm.permission.RECEIVE	692
android.permission.GET_ACCOUNTS	686
android.permission.VIBRATE	635
android.permission.ACCESS_WIFI_STATE	623
android.permission.READ_PHONE_STATE	611
com.android.vending.BILLING	424

3.3　Android アプリの自動リパッケージに対する耐性評価

本節では，Android アプリの自動リパッケージに対する耐性評価実験に関して，実験の目的，実験内容，実験結果について述べる．

3.3.1　実験目的

攻撃者が大量のリパッケージマルウェアを作成する際には，前章で述べたようなリパッケージの手法を自動化していることが予想される．しかし，実際に自動リパッケージを行う際の，リパッケージの成功率や，自動化にかかるコスト等については，これまで十分に検証が行われていない．また，リパッケージによって生成されたマルウェアは，元の正規アプリの機能を保持しているため，ユーザに継続的に利用され，長期間潜伏する恐れがあることから, その影響は大きいといえる．そこで本実験では，既存の正規アプリに対して，実際のリパッケージマルウェアにおいて用いられている自動リパッケージ手法により検証用コードを自動挿入し，(1) 挿入した検証用コードが正常に動作するかどうか，(2) リパッケージ済みアプリの動作が元のアプリから変化するかどうか，という 2 点についての検証を行うことを目的とする．

3.3.2　評価対象の正規アプリ

ダウンロード数等の観点で人気が高いことが予想される 1612 種類の正規アプリを 2015 年 2 月に Web 上から収集し，これを対象として自動リパッケージの耐性評価実験を行った．表 3.1 は収集した正規アプリが利用していたパーミッションの中から，利用される頻度が高かったものの上位 10 個を抜き出したものである．表の上位 2 つのパーミッションはアプリがインターネットに接続する際に要求されるパーミッションであり，今回評価対象とした正規アプリのほとんどが，このパーミッションを利用していた事がわかる．

表 3.2　静的解析対象のリパッケージマルウェア

検体 No.	検知名（AV ベンダ名）	MD5
No.1	Andr/Ksapp-A (Sophos)	97885f41713e7cc888567438626b0711
No.2	Android.Geinimi（Symantec）	90c05039fd16b78a7d4e7aaf803afaaa
No.3	Android/AndroRAT (AVG)	f64f38cdd6d12f10b1f9ee4bfb46ffc9

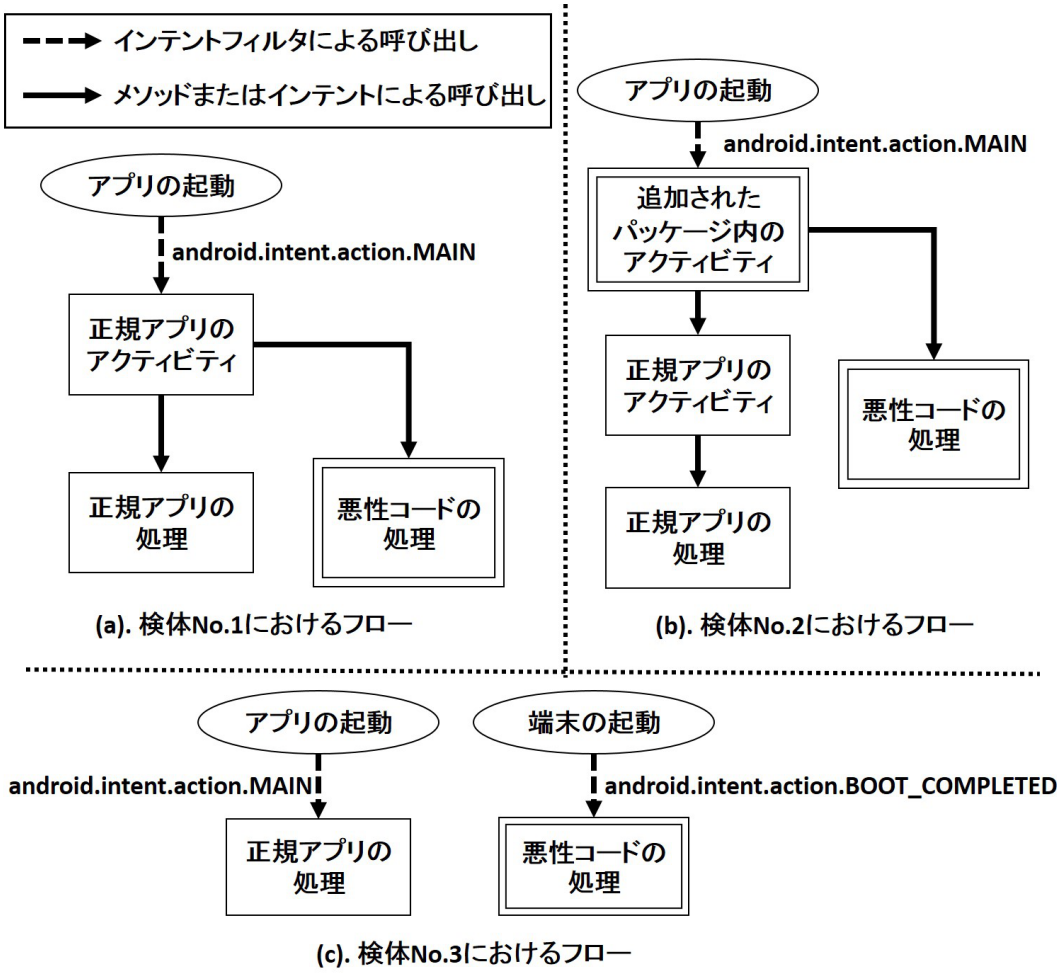

図 3.2　検体 No.1, 2, 3 における悪性コードが呼び出されるフロー

3.3.3　リパッケージマルウェアの解析

実際のマルウェアにおけるリパッケージ手法を調査するため，過去に流行した 3 種類のリパッケージマルウェアに対して静的解析を行った（表 3.2）．

解析の結果，全ての検体において，悪性パッケージの追加，AndroidManifest.xml の改変が見られた．ま

た，検体 No.1 においては，リソースファイルの追加や，正規アプリに含まれていたコードに対する改変も見られた．一方で，それぞれの検体において，追加された悪性コードの呼び出され方が異なっていた．図 3.2 に各検体における悪性パッケージが呼び出されるまでのフローを示す．検体 No.1 においては，挿入された悪性コードは正規アプリのアクティビティの中から呼び出されていたのに対して，検体 No.2 においては，”android.intent.action.MAIN”のインテントによって悪性コードが呼び出され，その後悪性コード内で元の正規アプリのアクティビティを明示的インテントによって呼び出していた．また，検体 No.3 においては，アプリの起動時には悪性コードが呼び出されず，端末が起動した際に発行される”android.intent.antion.BOOT_COMPLETED”のインテントによって悪性コードが呼び出されていた．

上記の内容から，それぞれの検体において以下のような手法で悪性コードの挿入が行われた事が予想される．

手法 1（検体 No.1）

1. 悪性パッケージの追加
2. ndroidManifest.xml 内にパーミッション，コンポーネントを追加
3. ”android.intent.action.MAIN”のインテントフィルタが設定されているアクティビティを特定
4. 該当アクティビティの onCreate メソッドの先頭部分に悪性パッケージ内のコードを呼び出すためのコードを挿入

手法 2（検体 No.2）

1. 悪性パッケージの追加
2. AndroidManifest.xml 内にパーミッション，コンポーネントを追加
3. "android.intent.action.MAIN”のインテントフィルタが設定されているアクティビティを特定
4. 該当アクティビティのインテントフィルタを削除
5. "android.intent.action.MAIN”のインテントフィルタを，追加した悪性パッケージ内のアクティビティに設定
6. 悪性コード内に，手順 3 において特定したアクティビティを明示的インテントによって呼び出す処理を追加

手法 3（検体 No.3）

1. 悪性パッケージの追加
2. AndroidManifest.xml 内にパーミッション，コンポーネント（android.intent.antion.BOOT_COMPLETED のインテントフィルタを設定済み）を追加

3.3.4 自動リパッケージ

以下の手順で自動リパッケージを行うためのスクリプトを作成した．

1. apktool を用いて自動リパッケージ対象の正規アプリをディスアセンブル
2. 検証用コードの挿入
3. apktool を用いて再ビルド

図 3.3　デフォルトのホーム画面が表示されている場合

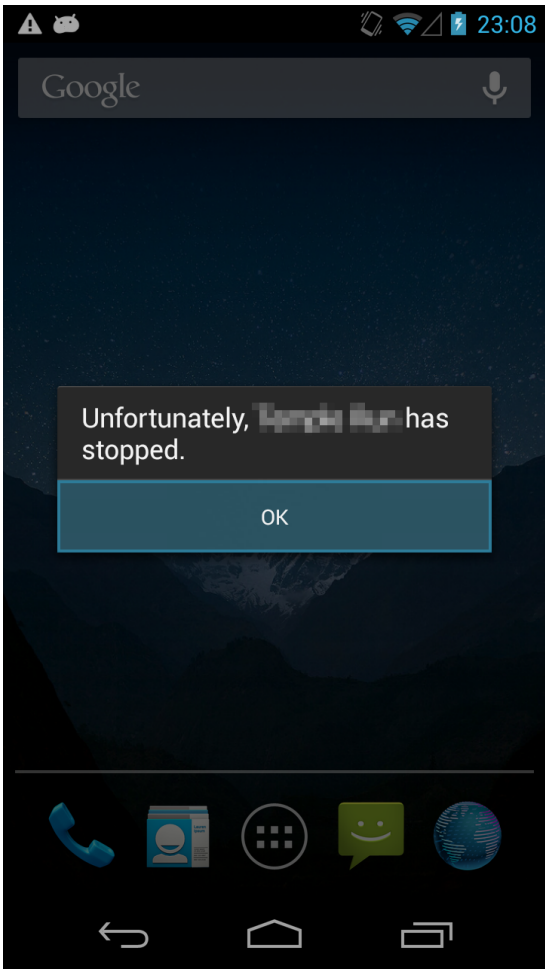

図 3.4　エラーメッセージが表示されている場合

4. jarsigner コマンドによる署名付与

今回の実験においては，手順 1，3 における正規アプリのディスアセンブル・再ビルドの際には apktool（version 2.0.0RC3）を用いた．また，手順 2 においては挿入する検証用コードとして外部に用意したサーバと通信を行う機能をもつコードを用意した．検証用コードの挿入は，前節において解説した 3 種類のリパッケージマルウェアによって用いられていた手法（手法 1，2，3）と同様の手順で行った．手法 1 において，onCreate メソッドの先頭部分に悪性パッケージ内のコードを呼び出すためのコードを挿入する際には，apktool によって出力された smali ファイルに対する文字列マッチングを用いた．具体的には起動時に呼び出されるアクティビティが実装されているクラスの smali コードの中で“->onCreate(Landroid/os/Bundle;)”の文字列が発見された場合，その次の行に悪性パッケージ内のコードを呼び出すための smali コードを挿入した．さらに，手順 4 における jarsigner コマンドによる署名付与の際には，独自に生成した署名生成鍵を用いた．

3.3.5 リパッケージ済み正規アプリの動的解析

リパッケージ元のアプリと，各手法によってリパッケージされたリパッケージ済みアプリを動的解析する事により，挿入した検証用コードが動作するかどうか，また，リパッケージによって元の正規アプリの機能に変化が生じていないかどうかを検証した．本実験で用いた動的解析の環境は以下の通りである．

- 端末：Galaxy Nexus GT-I9250
- OS：Android OS 4.2.1
- ネットワーク：WiFi にてインターネットへ接続

今回の実験においては，動的解析を行う環境として，Android OS 4.2.1 を搭載した実機を用いた．本実験で用いた自動リパッケージの手法は，OS のバージョンに依存しないため，異なるバージョンを用いた場合でも，本質的な実験結果は大きく変化しないと考えられるが，今後の Android OS のバージョンアップにおいて，フレームワークの仕様に大きな変化があった場合はこの限りではない．

解析対象のすべてのアプリに対して手動での詳細な動的解析を行う事はコストが高く，また，自動化された動的解析によって，アプリの持つ機能を網羅的に検証する事は難しいと考えられる．これらを踏まえ，今回の実験では，自動，及び手動での動的解析の両方を用いる事で，段階的な検証を行った．

自動で行われた動的解析の手順

1. adb [30](Android Debug Bridge) を用いて解析対象アプリのインストール・起動
2. 60 秒間待機した後に画面のスクリーンショットを撮影
3. (手法 3 の場合のみ)adb を用いて android.intent.action.BOOT_COMPLETED のインテントを発行し，検証用コードを起動
4. 外部サーバに対して，通信が行われたか判断
5. 対象アプリのアンインストール

まず，上記の解析手順を自動化することにより，すべてのアプリに対して，アプリが起動したかどうか，また，挿入された検証用コードが動作したかどうか，さらに，リパッケージ後にアプリの起動直後の画面の表示に変化が生じたかどうか，の 3 点について検証した．今回の実験における，アプリが正常に起動したかどうかの定義として，画面のスクリーンショットを用いた．具体的には，アプリの起動後の画面のスクリーンショットを目視にて確認し，図 3.3，図 3.4 のようにデフォルトのホーム画面や，エラーメッセージのダイアログが表示されていた場合を，アプリの起動の失敗，と定義した．動的解析中に，外部に用意したサーバに対して通信が発生したものについては，挿入した検証用が正常に動作したと判断した．さらに，リパッケージ前後の両アプリのスクリーンショットの比較を行い，アプリの起動直後の画面表示に変化が生じたかどうかを確認した．2 枚のスクリーンショットの画像の画素ごとの比較を行い，画素が 90% 以上一致していた場合は，アプリ起動直後の動作が変化しなかったと判断した．ここで画素の一致率に対して閾値を設定しているのは，日時や広告表示などといった，リパッケージの有無に関わらず，アプリの起動のタイミングによって画面上で変化する部分を無視するためである．画素の一致率が 90% 以下であった場合には，目視での確認を行いながらリパッケージ前後の両アプリを起動し，本質的な画面表示が変化していないかどうかを主観的に判断した．

次に，起動直後の画面の表示以外で，アプリの持つ機能に変化が生じていないかどうかを確認するために，手動での動的解析を行った．今回は実験の簡略化の為に，自動的な動的解析の結果において，アプリが正常に

表 3.3　正規アプリの動的解析結果

	アプリ数 *
収集した正規アプリ	1612 (100%)
端末へのインストールに成功	1549 (96.1%)
アプリの起動に成功	1520 (94.3%)

* 括弧内は収集した正規アプリに対する割合

表 3.4　正規アプリの動的解析結果

	アプリ数 *		
	手法 1	手法 2	手法 3
自動リパッケージ対象とした正規アプリ	1520（100%）		
apktool によるディスアセンブルに成功	1516（99.7%）		
検証用コードの挿入に成功	1405（92.4%）	1516（99..7%）	
再ビルド，jarsigner による署名付与に成功	1305（85.9%）	1379（90.7%）	1460（96.1%）

* 括弧内は自動リパッケージ対象とした正規アプリに対する割合

起動し，かつ検証用コードが動作し，さらにリパッケージ後に起動直後の画面の表示が変化しなかったものの中から，アプリ収集時に確認したユーザによるインストール数が 5000 万を超えていた 33 種類のアプリを手動での動的解析の対象とした．手順としては，まず，リパッケージ前のアプリを適切な操作を加えながら動的解析し，それぞれのアプリの持つ各種機能の動作を確認した．この際，アプリが持つ全ての機能を網羅的チェックすることは難しいため，通常使用の範囲内で利用される可能性が高いと思われる機能を優先的にチェックした．これらの機能を検査項目として，リパッケージ後のアプリに対しても同様に動的解析を行い，各種機能の動作に変化が生じていないかどうかを主観的に判断した．

3.4　実験結果

3.4.1　収集した正規アプリの動的解析

収集した正規アプリのリパッケージを行う前の動的解析の結果を表 3.3 に示す．収集した 1612 種類の正規アプリの内，最終的に 1520 種類のアプリが解析環境の端末上で起動に成功した．起動に失敗した 92 種類のアプリに関しては，リパッケージの前後のアプリの挙動を比較することができないため，自動リパッケージの対象から除外した．

3.4.2　自動リパッケージ

自動リパッケージにおける，ディスアセンブル，検証用コードの挿入，再ビルド，署名付与の各処理に成功したアプリの内訳を表 3.4 に示す．今回自動リパッケージ対象としたアプリの中には，そもそも apktool によるディスアセンブルに失敗するものが 4 種類存在した．また，検証用コードの挿入時には，手法 1 においての

表 3.5　リパッケージ済み正規アプリの自動での動的解析結果

		アプリ数		
		手法 1	手法 2	手法 3
自動リパッケージ対象とした正規アプリ		1520（100%）		
動的解析対象としたリパッケージ済みアプリ		1305（85.9%）	1379（90.7%）	1460（96.1%）
端末へのインストールに成功		1286（84.6%）	1359（89.4%）	1440（94.7%）
アプリの起動に成功		1194（78.6%）	1351（86.5%）	1411（92.8%）
外部サーバへの通信が発生		1206（79.3%）	1351（88.9%）	1415（93.1%）
アプリの起動時の画面表示に変化が生じたかどうか	画面に変化無し	1169（76.9%）	1290（84.9%）	1381（90.9%）
	画面に変化有り	136（8.9%）	89（5.9%）	79（5.2%）
外部サーバへ通信が発生し，尚且つ画面表示に変化が商生じなかったもの		1162（76.4%）	1286（84.6%）	1368（90.0%）

* 括弧内は自動リパッケージ対象とした正規アプリに対する割合

み，111 種類のアプリに対して処理が失敗した．これらに関しては，onCreate メソッドの先頭部分を特定する際に用いた文字列マッチングの結果，コード挿入場所を見つける事が出来なかった場合であった．さらに，検証用コードの挿入後，apktool によって再ビルドを行う際にエラーが発生し，処理に失敗する検体が，手法 1 においては 100 種類，手法 2 では 137 種類，手法 3 では 56 種類存在した．署名の付与については，すべての手法の全検体に対して，問題なく処理が成功した．各手法における結果を比較してみると，全ての処理に成功したアプリの割合は，手法 1 では 85.9% であったのに対して，手法 2 では約 90.7%，手法 3 では 96.1% と，手法 1 より高い成功率となった．　各手法において，全ての処理に成功したアプリに対して動的解析を行い，挙動を確認した．

3.4.3　リパッケージ済み正規アプリの自動での動的解析

リパッケージ済み正規アプリの自動的に行われた動的解析の結果を表 3.5 に示す．自動リパッケージの全ての処理に成功したアプリの中で，手法 1 では 1289 種類，手法 2 では 1359 種類，手法 3 では 1440 種類のものが，解析環境の端末へのインストールには成功したが，各種法において，19〜20 種類の検体は”INSTALL_PARSE_FAILED_MANIFEST_MALFORMED”のエラーが原因でインストールに失敗した．これは，インストール対象のアプリの AndroidManifest ファイルの形式が不正である場合に発生するエラーであり，リパッケージの処理の際に，AndroidManifest の改変に失敗している事が予想される．また，最終的に外部サーバへの通信が確認できたものが，手法 1 においては 1206 種類，手法 2 においては，1351 種類，手法 3 においては 1415 種類存在した．これらのアプリに関しては，挿入した検証用コードが正常に動作していると考えられる．更に，検証用コードは正常に動作したが，アプリの起動には失敗しているパターンが手法 1 では 24 種類，手法 2 では 42 種類，手法 3 では 22 種類のアプリにおいて確認された．これは，検証用コードが実行された後に，正規アプリコード部分にてエラーが発生し，アプリが強制終了している場合であると考えられる．

リパッケージ前後の，各アプリの挙動に関しては，各手法において 7 割以上が，自動リパッケージを行って

表 3.6　リパッケージ済みアプリの手動での動的解析結果

	アプリ数 *
手動での動的解析対象としたアプリ	33 (100%)
全ての検査項目において，アプリの機能が保持されていたもの	29 (87.9%)
一部の検査項目において，アプリの機能に変化が生じていたもの	4 (12.1%)

* 括弧内は手動での動的解析の対象としたアプリに対する割合

も，起動後の画面表示に変化が無いことがわかる．最終的に，外部サーバへ通信が発生し，尚且つリパッケージ後に起動後の画面表示の機能に変化が無かったものが，手法 1 において 76.4%，手法 2 においては 84.6%，手法 3 においては 90.0% 存在した．

いずれの手法を用いても，挿入した検証コードが動作し，尚且つ画面の表示に変化が無かったアプリが 1078 種類存在した．今回は，それらの中から，ユーザによるインストール数が 5000 万回以上であった，33 種類のアプリに対して，手動での動的解析を行った．

表 3.7　手動での動的解析の際の検査項目，および検査結果（アプリの機能が保持された場合，抜粋）

アプリの種類（インストール数）	検査項目としたアプリの機能	リパッケージ後の動作（○：変化無し，-：変化有り）			備考
		手法 1	手法 2	手法 3	
ゲームアプリ A（5000 万〜1 億）	ランチャーからアプリを起動	○	○	○	
	基本的な UI の操作	○	○	○	
	オフラインプレイ	○	○	○	
	オンラインプレイ	○	○	○	
ゲームアプリ B（1 億〜5 億）	ランチャーからアプリを起動	○	○	○	
	基本的な UI の操作	○	○	○	
	ゲームプレイ	○	○	○	
	SNS 連携	○	○	○	
Web ブラウザアプリ（5000 万〜1 億）	ランチャーからアプリを起動	○	○	○	
	基本的な UI の操作	○	○	○	
	Web ページ閲覧	○	○	○	
SNS アプリ（1 億〜5 億）	ランチャーからアプリを起動	○	○	○	
	基本的な UI の操作	○	○	○	
	ログイン	○	○	○	
	投稿	○	○	○	

表 3.8　手動での動的解析の際の検査項目，および検査結果（アプリの機能に変化が生じた場合，抜粋）

アプリの種類（インストール数）	検査項目としたアプリの機能	リパッケージ後の動作（○：変化無し，-：変化有り）			備考
		手法 1	手法 2	手法 3	
Android 用 AV ソフト（1 億〜5 億）	ランチャーからアプリを起動	○	○	○	
	基本的な UI の操作	○	○	○	
	AV スキャン	○	○	○	
	ウイルス定義更新	-	-	-	ウイルス定義の更新に失敗
天気情報アプリ（5000 万〜1 億）	ランチャーからアプリを起動	○	○	○	
	基本的な UI の操作	○	○	○	
	天気情報表示	○	○	○	
	地図情報の表示	-	-	-	地図の表示に失敗
フォトエディター（1 億〜5 億）	ランチャーからアプリを起動	○	○	○	
	基本的な UI の操作	○	○	○	
	画像編集機能	○	○	○	
	ソーシャル機能：ログイン	-	-	-	サーバとの通信に失敗
無料通話・メッセンジャーアプリ（1 億〜5 億）	ランチャーからアプリを起動	-	-	-	起動時にアプリのアップデートに失敗

3.4.4 リパッケージ済み正規アプリの手動での動的解析

手動での動的解析の結果を表 3.6 に示す．手動での動的解析を行ったアプリのうち 87.7 %のものは，すべての検査項目において，元の正規アプリの機能が保持されていた．一方で，4 種類においては，一部の機能が保持さておらず，結果としてアプリを本来の目的において使用することが出来ない状態となっていた．4 種類のアプリにおいて保持されなかった機能は，アップデート機能や，サービスへのログイン機能など，いずれも外部のサーバとの通信が発生する機能であった．

なお，手動での動的解析の際の検査項目，及び検査結果について，アプリの機能が保持されなかった 4 種類，及びそれ以外から抜粋した 4 種類のものを表 3.7，表 3.8 に示す．

3.5 考察

3.5.1 正規アプリの自動リパッケージへの耐性

今回の検証では，自動リパッケージによって長期的に感染を続けるようなマルウェアを作成する攻撃者を想定している．攻撃者の視点から考えると，元の正規のアプリと同じ機能を提供することでユーザに長期的にアプリを使用させつつ，その裏で挿入した悪性コードを定常的に動作させ続ける事を目的としていると予想される．これらを踏まえると，(1) 挿入された悪性コードが動作しない場合，もしくは (2) 挿入された悪性コードは動作するが，元の正規アプリの機能が損なわれている場合のどちらかに当てはまるアプリが，今回想定したような目的で行われる自動リパッケージに対して耐性があると言える．

また，悪性コードが動作する期間の長さから考えると，悪性コードが全く動作しない (1) の条件を満たすものが最も耐性が強く，(2) の条件を満たすアプリの中でも，正規アプリの機能が損なわれる事により，より早いタイミングで悪性コードが動作しなくなるものの方が耐性が強いと言える．加えて，上記の (1)，(2) の条件全てに当てはまらないアプリについては，今回想定する攻撃に対しては，全く耐性が無いと言える．

前章の実験結果を見てみると，用いた手法によって結果に差があるものの，全体の 7 割から 9 割のアプリにおいて，挿入した検証用コードが正常に動作し，尚且つ起動後の画面の表示に変化が生じていない事が分かる．これらのアプリについては，上記の (1) には当てはまらず，また，少なくとも起動直後の画面表示に関わる機能は損なわれていないため，そもそもリパッケージ処理に失敗したアプリや，起動に失敗するようになったアプリと比較して，自動リパッケージへの耐性が低いと言える．さらに，それらの中から抽出した 33 種類のアプリのうち，29 種類については，手動での動的解析の結果から，元のアプリの機能が保持されている事が分かる．これらの 29 種類については，今回の検証の範囲内では，上記の (1)，(2) のどちらの条件にも当てはまらない為，自動リパッケージに対する耐性が無いといえる．以上より，現状の Android アプリの多くは自動リパッケージへの耐性が不十分であり，リパッケージ元の正規アプリと同様に動作し，その裏で悪意のあるコードを実行するマルウェアが大量に作成される可能性が十分に考えられる．

3.5.2 アプリの動的解析手法について

今回は実験の簡略化の為に，アプリの動的解析の際に，アプリが正常に起動したかどうかの判断や，リパッケージによって元の正規アプリから起動直後の動作が変化しないかどうかを，スクリーンショットを用いた機械的な方法で判断した．一方，起動後に画面の表示に変化は生じるが，画面の表示以外の内部処理でエラーが

表 3.9　リパッケージ処理中に発生したエラーの内訳

	アプリ数 *		
	手法 1	手法 2	手法 3
リパッケージ処理中にエラーが発生した全アプリ	215（100%）	141（100%）	60（100%）
（a-1）apktool によるディスアセンブルに失敗	4（1.9%）	4（2.8%）	4（6.7%）
（a-2）検証用コードの挿入に失敗	111（51.6%）	0（0%）	0（0%）
（a-3）apktool による再ビルドに失敗	100（46.5%）	137（97.2%）	56（93.3%）

* 括弧内はリパッケージ処理中にエラーが発生した全アプリに対する割合

表 3.10　動的解析時に発生したエラーの内訳

	アプリ数 *		
	手法 1	手法 2	手法 3
動的解析時にエラーが発生した全アプリ	143（100%）	93（100%）	92（100%）
（b-1）端末へのインストールに失敗	19（13.3%）	20（21.5%）	20（21.7%）
（b-2）アプリの起動に失敗	93（65.0%）	44（47.3%）	29（31.5%）
（b-3）検証用コードが動作せず【b-1 以外】	80（55.9%）	8（8.6%）	25（27.2%）
（b-4）アプリの挙動が変化【b-1，b-2 以外】	24（16.8%）	26（28.0%）	30（32.6%）

* 括弧内は動的解析中にエラーが発生した全アプリに対する割合

発生し実際には正常にアプリが起動していない場合が考えられる．さらに，ユーザによるタッチ等のイベントによってアプリの内部処理に分岐が生じる場合や，後述するリパッケージ対策がアプリの起動直後以外の場所に施されている場合も考えられる．上記を考慮することで，より細かい粒度での自動リパッケージへの耐性を検証することが可能であるが，一方で実験のコストから考えると，自動化可能で尚且つより詳細にアプリの挙動を把握可能である動的解析手法が必須であると考えられる．具体的には，デバッギング技術等を用いてより詳細なログを取得する手法や，動的解析時のコードカバレッジを向上させるような手法の組み合わせなどが考えられる．

3.5.3　自動リパッケージに失敗したアプリについて

リパッケージにおける各処理や，コード挿入後の動的解析時において何らかのエラーが発生した検体が存在した．表 3.9，表 3.10 にそれぞれのエラーが発生したパターンと検体数についてまとめる．

a. アプリのリパッケージ処理（ディスアセンブル〜署名付与）で発生したエラーについて

- **(a-1)** apktool によるディスアセンブルに失敗したケースについて調査した結果，apktool のバグに起因している事が判明した．これらについては，今後の apktool のアップデートによって処理に成功するようになる可能性がある．
- **(a-2)** 手法 1 においてコード挿入位置が特定出来なかったケースについては，マッチングに用いた文字

列が不適切であった場合，つまり特定の実装パターンにおいては対象とした文字列が smali コード内に現れない場合があった為であると考えられる．また，今回用いた文字列検索を，smali コードを厳密に解釈して挿入位置を決定するアルゴリズムに変更することで，より多くの検体に対して処理が成功することが予想されるが，一方で，そのような高度な処理を自動化するのはコストが高いため，成功率とコストのトレードオフになることが予想される．

- **(a-3)** apktool による検証用コード挿入後の再ビルド時にエラーが発生したケースについては，様々な原因が考えられるが，多くの場合で，挿入されたコードが原因で，smali コードのフォーマットに不整合が生じたために，dex 形式へと再ビルドすることが出来なかったためであると考えられる．

b. **動的解析時に発生したエラーについて**

- **(b-1)** リパッケージ後に端末へのインストールに失敗するようになった検体について調査した結果，リパッケージ処理において AndroidManifest の改変に失敗しており，結果として AndroidManifest が不正な状態で再ビルドが行われていたことが判明した．これらのエラーはリパッケージを行うスクリプトの改良によって改善されると考えられる．
- **(b-2)** リパッケージ後にアプリの起動に失敗するようになったケースについては，アプリの起動時にエラーが発生し，強制終了している場合と，エラーは発生していないが，アプリを起動させてもデフォルトのホーム画面から変化が生じない 2 つのパターンが存在した．
- **(b-3)** 挿入した検証用コードが動作しなかったケースについて，検証用コードが動作せず，尚且つアプリの起動に失敗している場合が，手法 1 では 69 種類，手法 2 では 2 種類，手法 3 では，7 種類のアプリにおいて確認された．これらについては，処理が検証用コードに到達する前に，エラーが発生し，アプリが強制終了したパターンが原因として考えられる．各手法において，上記のようなパターンであったアプリ数に差が見られるが，これは，それぞれの手法における検証用コードの呼び出され方の違いに起因している事が予想される．
- **(b-4)** アプリのインストール，及び起動には成功するが，リパッケージ前と比較してアプリの動作に変化が生じたケースについては，起動時に外部に用意されたサーバと通信するタイプのアプリにおいて，通信が正常に行われなくなるパターンや，リパッケージ後にのみポップアップによる通知等が現れるパターンなどが存在した．

上記のエラーのうち，(a-1)，(b-1) については，リパッケージ処理の不備が主なエラーの原因であると考えられるが，一方で，(a-2)，(a-3)，(b-2)〜(b-4) については，アプリになんらかのリパッケージ対策がなされていた為に処理に失敗した可能性も考えられる．特に (b-4) については，5 種類のアプリにおいて明確にリパッケージ対策がなされている事を確認した．（3.5.6 節を参照）

3.5.4 検証用コードの挿入方法による自動リパッケージの成功率の差異

今回の実験では，検証用コードの挿入時 3 つの手法を用いたが，各種法ごとに自動リパッケージの成功率に差が見られた．表 3.11 は各手法において自動リパッケージの際に行った処理の内容を比較したものである．最も成功率が低かった手法 1 は，元の正規アプリに含まれていた実行コードに対する改変を行っており，これが前節における (a-2)，(a-3)，(b-2) のようなエラーの発生数を増加させる要因になっている事が予想される．また，手法 2 は全手法の中で検証用コードが呼び出されるタイミングが最も早いため，(b-3) のようなエラー

表 3.11　各種法において自動リパッケージの際に行った処理の比較

処理内容		手法 1	手法 2	手法 3
元の正規アプリに含まれていた実行コード（smali ファイル）の改変		○	-	-
AndroidManifest の改変	既存コンポーネントの定義の改変	-	○	-
	コンポーネント・パーミッションの追加	○	○	○
新規パッケージの追加		○	○	○

が発生しにくいと言える．さらに，手法 3 では，パッケージの追加，コンポーネント・パーミッション定義の追加のみを行っており，他の手法と比較して，元の正規アプリ含まれていたコード・リソースへの改変が少ないため，全体として，処理に失敗する検体数が少なかったと考えられる．

3.5.5　自動リパッケージの手法と挿入されたコードのエントリポイントについて

今回の実験では，過去に流行した 3 種類のリパッケージマルウェアの中で実際に用いられていたリパッケージ手法を模擬したが，それぞれの手法は，挿入されたコードの呼び出され方が異なっていた．自動化が可能なリパッケージの手法は，挿入されるコードが何をトリガーとして実行されるか，という観点からいくつかのパターンに分類可能であると考えられる．

まず，ユーザによるアプリの起動をトリガーとして挿入されたコードが実行される手法を考える．通常，アプリが起動する際には，AndroidManifest 内で android.action.MAIN のインテントフィルタが指定されたアクティビティが呼び出される．アクティビティ起動後の処理は各々のアプリの実装に依存してしまうため，自動化が可能な方法としては，(a) 起動直後に呼び出される正規アプリのアクティビティを機械的に改ざんし，悪性コードのエントリポイントを挿入する方法，(b)android.action.MAIN の受け取り先自体を変更し，任意の処理を行った後に，明示的インテント等を用いて元々呼び出される筈であった正規アプリのアクティビティを呼び出す方法，の 2 種類が考えられる．本実験における手法 1，手法 2 はそれぞれ (a)，(b) に分類される．　アプリの起動以外でコード実行のトリガーに成りうるものとして，ブロードキャストインテントが考えられる．Android では，例えば端末の再起動や，SMS の受信など，端末上で発生したイベントに応じてアプリが任意のコードを実行するための仕組みとして，ブロードキャストインテントが用いられる．ブロードキャストインテントを利用した自動化可能なリパッケージの手法としては，(c)AndroidManifest 内に新たにブロードキャストインテントを受け取るための BroadcastReceiver を定義し，そこをエントリポイントとして任意の処理を実行する方法が考えられる．本実験における手法 3 が上記 (c) に分類される．　リパッケージの手法が自動化可能であるかどうかを考える際には，挿入されるコードを呼び出すためのエントリポイントに当たる部分を機械的に挿入することができるか，という問題に帰着する事が予想される．よって今回利用したアプリの起動や，ブロードキャストキャストインテント以外のエントリポイントであっても機械的な処理で挿入可能であれば，自動リパッケージに利用される可能性があると言える．

3.5.6　リパッケージ対策が施されていたアプリについて

今回の評価対象とした正規アプリの中で，リパッケージへの対策がなされていたと思われるものが見受けられた為，以下にまとめる．

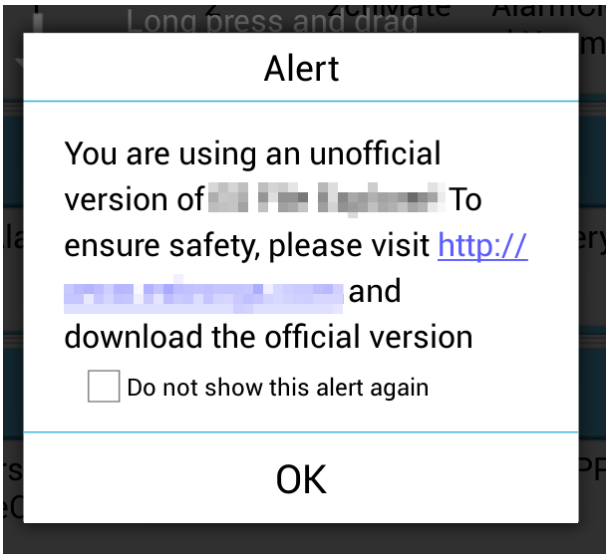

図 3.5　リパッケージ済みアプリの動的解析時に表示されたダイアログ

リパッケージ後のアプリを動的解析した際に，図 3.5 のようなダイアログが表示されるものが存在した．これは，アプリが自身の改ざんを検知して，ユーザに対して注意喚起を行っていると考えられる．今回の実験中に，5 種類のアプリが，同様のポップアップ表示によってユーザに対し注意喚起を行っているのを確認した．このような改ざん検知機能によって，ユーザはマルウェアの存在に気付きやすくなり，端末内へ長期間潜伏するようなタイプのマルウェアの被害を軽減することができると考えられる．改ざんを検知するための具体的な方法としては，アプリに施された署名情報のチェックや，実行コードやリソースファイルのハッシュ値のチェックなどが考えられる．

3.5.7　耐タンパー化によるリパッケージ対策

自動リパッケージへの対策としては，前節で例示した改ざん検知等の一般的な耐タンパー技術が有効であると考えられる．Android に適用可能な耐タンパー技術として，論文 [63] にて提案されている手法が挙げられる．論文 [63] 内では，Android NDK [31] を用いて Android 上で自己書き換え型の耐タンパー技術を実現している．Android NDK において，実行コードは Java では無く，より解析が困難であるネイティブコードにて記述されるため，前述したようなアプリの改ざん検知機能を Android NDK を用いて実装することで，攻撃者による自動リパッケージを，より困難にすることが可能である．

3.5.8　リパッケージ対策の今後の展望

上記のような対策が考えられる一方で，技術レベルの高い攻撃者が手動でリパッケージを行った場合には，Android におけるリバースエンジニアリングの容易さなどから，それらを完全に防ぐことは難しいと考えられる．しかし，一般的に，個々のアプリに対応した高度なリパッケージの処理を自動化することは困難であるため，今回検証を行ったような自動リパッケージを防ぐための対策を行うだけでも十分に被害を軽減できると考えられる．多くの正規アプリにおいて，自動リパッケージの対策がなされていない現状を踏まえると，例え

ば，自動的にアプリに対して耐タンパー化を施してくれるようなツールや，耐タンパー性を高める開発方法のマニュアル化などといった，セキュリティに関する知識が十分でないアプリ開発者でも，「自動リパッケージを防ぐための機能」をアプリに容易に付与できる対策技術が必要であると言える．

一方で，上記のような対策技術が一般化すると，これを自動的に回避する攻撃が現れ，対策が陳腐化し，いわゆる「いたちごっこ」の状態に陥ってしまう事が予想される．しかし，攻撃側は自動リパッケージを行う対象のアプリをブラックボックスとして扱わなければならないのに対して，対策を行う側 (開発側) は，その内容やソースコードを所持しており，守る側に有利ある．今後は，そういった優位性を活かす事で，陳腐化しにくい自動リパッケージ対策を検討していく必要がある．

3.5.9　正規アプリが要求するパーミッション

今回評価対象とした正規アプリのほとんどが，ネットワーク接続のためのパーミッションを要求していた．また，GET_ACCOUNTS や READ_PHONE_STATE などといった，ユーザの個人情報にアクセスする際に必要とされるパーミッションを合わせて要求しているものも見受けられた．例えば，電話帳アプリに対して連絡先情報を漏洩する悪性コードを挿入する場合や，地図アプリに対して位置情報を漏洩する悪性コードを挿入する場合などのように，攻撃者が，悪性コードに実装した機能に応じてリパッケージ対象の正規アプリを選ぶことで，新たにパーミッションを加えること無くリパッケージが行われる可能性がある．これらのリパッケージマルウェアは，ユーザがパーミッションの確認を行っても不審に見えないため，より危険性が高いと言える．

3.5.10　Android マルウェアの流通・流行に関して

攻撃者が自動リパッケージを行って大量のリパッケージマルウェアを作成した後に，どのような経路でユーザの端末へ感染させるかについても今後調査が必要である．攻撃者は多くの場合，アプリ掲載時の審査が不十分なサードパーティマーケットにおいてリパッケージマルウェアを配布していると考えられる．先行研究 [2] では，マーケットプレイス上でリパッケージマルウェアを検出する手法が提案されており，今後，リパッケージマルウェアに対する対策を考える上で，前述した耐タンパー化のような，各アプリの作成者が主体となって行う対策に加えて，マーケットを運営しているキャリアやベンダが主体となって行う対策も組み合わせていくことが必要と考えられる．

3.6　まとめと今後の課題

本章では，Android アプリケーションの自動リパッケージに対する耐性評価実験を行った．その結果，用いた手法により成功率に差が見られたものの，評価対象とした既存の正規アプリの内，7〜9 割のものに対して，起動時の画面表示機能に影響を与えることなく検証用コードを挿入することが可能であることを示した．更に，多くのユーザによって利用されている，人気のアプリにおいて，リパッケージを行ってもアプリの持つ基本的な機能に変化が生じない例を確認した．今回挿入した検証用コードを，悪性な動作を行うコードに置き換える事は容易であるため，現状の Android アプリケーションの多くは，攻撃者によって容易にリパッケージが可能であり，リパッケージ元の正規アプリと同様に動作し，その裏で悪性コードを実行することで，長期的に感染を続けるようなマルウェアが大量に作成される危険性があると言える．

自動リパッケージへの対策としては，アプリの改ざんを検知する機能等が有効であると考えられる．一方，攻撃者が高度な技術を用いてリパッケージを行った際には，それらを完全に防ぐことは難しいが，自動的なリ

パッケージを防ぐ対策は攻撃者のコストを増加させる点で被害を軽減する効果が期待できる．

今後の課題としては，陳腐化しにくい自動リパッケージ対策手法の検討や，アプリマーケットと連携したリパッケージ対策などが考えられる．また，多くの正規アプリにおいてリパッケージ対策が不十分である現状を踏まえ，セキュリティに関する知識が十分でないアプリ開発者でも容易に対策を行う事が可能な技術も合わせて検討する必要がある．これを踏まえ，4 章では，個々の開発者に依存せず Android アプリに対して自動的に改ざん対策を付与するシステムの構成方法について提案する．

第 4 章

Android アプリの改ざん対策技術の方式検討

4.1 はじめに

Android アプリは，オープンな仕様や多数の解析ツールが公開されている事などから，リバースエンジニアリング技術を用いたアプリの改ざんが容易である．攻撃者は，マーケットから入手した正規のアプリを改ざんすることで，マルウェアや海賊版アプリを作成する [13] [44]．アプリ改ざんによって作成されるマルウェアは，見かけ上は元の正規のアプリのように動作するが，その裏で悪性な活動を行う．この場合，ユーザはマルウェアの感染に気付きにくく，特に Bot や RAT などといった，長期感染を目的としたマルウェアによる被害が拡大する恐れがある．同様に，攻撃者は正規アプリに含まれる広告モジュールの改ざんや新たな広告モジュールの追加により，本来アプリ開発者が得るはずであった広告収入を詐取する攻撃も行う．FireEye 社による調査によれば，Android を標的としたアドウェア “kemoge” はアプリ改ざんによる不正な広告モジュールの埋め込みによって作成され，20 カ国以上で被害が確認された事が報告されている [23]．

Android においては，これまで PC の領域で用いられてきたコードサイニング技術のような，OS によってサポートされたアプリの作成者を検証する仕組みが存在しない．また，そのような仕組みの導入は，既存の OS の改変が必要であるため，後方互換性の観点から困難である．よって，現在の環境にも適用が容易な，アプリ自身に対する改ざん対策の付与が重要である．

アプリに改ざん対策を付与する際には，例えば，難読化 [21] [58] やアンチデバッグ [15]，自己改ざん検知 [40] [48] などの複数の技術を組み合わせることで，攻撃者が改ざんにかかるコストを増加させるアプローチが取られる．攻撃者は，改ざんにかかるコストが得られる利益やリスクに見合っている場合にのみ攻撃を行うと考えられるため，このような攻撃コストを増加させる対策によって被害の軽減が期待できる．

一方，このような対策は，開発者が自発的に対策付与を行う必要があり，実装の際にはアプリ改ざんに関する知識や実装能力が要求される．このため，個々の開発者のセキュリティに対する意識や実装スキルに依存して，そもそも対策が付与されない場合や，対策の強度が弱く容易に回避されてしまう場合がある．実際，3 章で行った評価の結果では，ダウンロード数が数千万回を越えるような人気アプリを多数含んだデータセットのうち 9 割以上のアプリに対して全自動で改ざんが成功する事が確認された．また，簡易的な改ざん対策を自動的に無効化するツールが公開された事例も存在する [39]．以上から，現状ではアプリに対して付与する改ざん対策の効果は限定的であるといえる．

本章では個々のアプリ開発者に依存せず，アプリに対して堅牢な改ざん対策を自動的に付与する事で，攻撃者が改ざんを行う際のコストを増加させるシステムの構成方法について検討する．

なお，本章で検討するアプリの改ざん対策技術は，アプリ自身の堅牢化を目的とした対策であり，例えばア

プリマーケットやエンドユーザの立場から改ざんによって作成されるモバイルマルウェアを検知する既存技術と補完関係にある．そのため，これらの手法を組み合わせる事で改ざんに起因するマルウェアに対する多層的な防御が可能となる．

4.2 要素技術

アプリに改ざん対策を行う上で，難読化や自己改ざん検知などのアプリを堅牢化する複数の技術を組み合わせることで，攻撃者が改ざんにかかるコストを増加させる事が有効である．本節では，アプリの堅牢化を行う上での代表的な要素技術について説明を行う．なお，本節にて説明を行う要素技術はアプリの堅牢化を目的とした全ての技術を網羅するものではなく，一部の技術は複数の要素技術に分類される場合がある点に注意されたい．

アプリの難読化

難読化とは，実行コードを解析が困難かつ機能的には等価な状態へ変換する技術である．主に静的解析を妨害するために用いられる．例としては，Android SDK に付属する ProGuard [21] による，変数名等のシンボル情報を無意味な文字列に置き換える手法が挙げられる．また，より高度な難読化手法としては，アプリの制御フローを機能的に等価かつ複雑な形に変形する制御フローの難読化や，アプリおける定数文字列を暗号化し対応する復号ルーチンを挿入する文字列の暗号化などが挙げられる．また，Android においては Java によって実装され Dalvik 仮想マシンに実行されるバイトコードの他に，C/C++ などによって実装されるネイティブコードの二種類のコードが実行可能である．一般的に，逆コンパイルが容易なバイトコードと比較して，ネイティブコードの方が解析の難易度が高いため，アプリの持つ機能をネイティブコードとして実装する事も広義の難読化と言える．

アンチデバッグ

アンチデバッグとは解析環境やツールを検出し，それらの処理を妨害する技術である．主に動的解析を妨害する為に用いられる．例としては，デバッガや仮想環境を検知し，それらが検知された際には，本来の動作とは異なる処理を行ったり，アプリの動作自体を停止したりする事で解析を妨害する．

実行コードの暗号化・パッキング

実行コードの暗号化とは，アプリに含まれる実行コードを予め暗号化しておき，アプリの実行時に必要に応じて動的に復号し実行する技術である．このような技術はパッキングとも呼ばれる．攻撃者が復号された本来の実行コードを入手するには，アプリの実行中にメモリやファイルシステム上に復号された実行コードを取得するか，アプリに含まれる復号ルーチンを解析し，それらの処理を模して暗号化された実行コードを復号する必要がある．実行コードの暗号化によるアプリの堅牢化の強度は処理の性質上，復号ルーチン解析の容易性や，動的解析時の復号された実行コード抽出の容易性に依存する．そのため，例えば難読化手法との組み合わせにより復号ルーチンの解析を困難にしたり，アンチデバッグ技術との組み合わせにより，動的解析時の実行コード抽出を困難にする対策との組み合わせが効果的である．

自己改ざん検知

自己改ざん検知とは，保護対象のアプリが自身の完全性検証を行い，改ざんを検知する技術である．改ざんが検知された際には，アプリの動作停止や，利用者への警告表示等を行い，改ざんされたアプリがユーザによって利用され続けないよう対応を行う．自己改ざん検知を行う方法には様々な手段があり得るが，典型的な例としては，アプリに付与されている署名の検証や，実行コードのハッシュ値の検証などが挙げられる．アプリに自己改ざん検知機能が付与されている場合，攻撃者は改ざんによって自己改ざん検知機能自体の無効化を試みると考えられる．そのため，自己改ざん検知機能の堅牢性を増加させる上では，難読化やアンチデバッグ技術との組み合わせにより，当該機能を実装した箇所が容易に特定できないような対策を行う事が効果的である．

4.3 アプリの改ざん防止に関する既存技術

本節では，アプリの改ざん防止を目的とした既存の学術研究，市中製品について説明する．

4.3.1 学術研究

アプリの改ざん防止を目的とした主な学術研究について説明する．Protsenko らは，ネイティブコードを利用した実行コードの難読化，及び自己改ざん検知手法を提案している [48]．この手法では，予めアプリのバイトコードをクラス単位で暗号化しておき，実行時にはネイティブコード内に実装された復号ルーチンが必要に応じてバイトコードの復号，及び完全性の検証を行う．

Divilar [58] はオペコードのランダム化，及び仮想インタプリタの自動生成による難読化手法である．この手法では，実行コードのオペコードをランダムな値に変換しておき，実行時には変換ルールに基づいて自動生成された仮想的なインタプリタがオペコードを逆変換しながら読み込む．これにより，既存ツールを用いた静的／動的解析が無効化される．

SSN [40] は自己改ざん検知機能をアプリに対して自動的に付与する手法である．アプリのソースコードに自己改ざん検知コードをランダムに多数挿入することで，攻撃者による検知機構の無効化を困難にしている．また，非決定性プログラミングのコンセプトを用いており，改ざんが検知された際に確率的な対応を行うことで，動的解析によるアプリのデバッグを困難にしている．

Protsenko らの手法，及び Divilar は，OS の実装に強く依存した手法であるため，手法適用によりアプリの互換性が失われ，Android 5.0 以降の環境で動作しなくなる問題がある．SSN は互換性の観点から優れている一方で，手法の入力データとしてアプリのソースコードを必要とする為，入力としてアプリの本体しか得られない状況においては手法を適用できない問題がある．

4.3.2 市中製品

アプリの改ざん防止を目的とした主な市中製品について説明する．DashO [49] は PreEmptive Solutions 社によって販売されている，Android/Java プログラム向けの難読化ツールである．開発者は，アプリのビルドプロセスの中に DashO による処理を組み込む事で，難読化や自己改ざん検知機能が付与されたアプリを作成可能となる．具体的な処理内容はブラックボックスであるが，公式サイトに記載されている機能一覧から，シンボル情報や制御フローの難読化，自己改ざん検知コードの追加などを行っている事が予想される．また，自己

改ざん検知コードの追加にあたっては，デベロッパがソースコード内の関数にアノテーションを付与する必要がある事がマニュアルに記載されており，効果的にアプリを保護するためにはデベロッパによる設定の調整が必要であると考えられる．

CrackProof for Android [62] は DNP ハイパーテック社によって販売されている，Android アプリの耐タンパ化サービスである．開発者がアプリをクラウドサーバ上にアップロードすることで，自動的にアプリに対して耐タンパ化処理が行われる．公式サイトに記載のある機能一覧から，自己改ざん検知，及びアンチデバッグ機能の追加，実行コードの暗号化などの処理を行っている事が予想される．

4.3.3 既存技術の比較

本節では 4.3.1 節および 4.3.2 節にて説明を行った既存技術を各手法における入力データ，対策に用いられている技術，対策の付与によるオーバヘッド，対策を付与したアプリの後方互換性の観点から比較した結果について述べる．また，表 4.1 に比較結果の概要を示す．

アプリの改ざん防止を目的とした既存技術には，アプリ本体である APK ファイルを入力とする場合と，アプリのソースコードを入力とする場合が存在する．ソースコードを入力とする技術は，アプリ開発者，もしくは開発者にソースコードの提出を義務付けているアプリマーケットなどの立場でしか利用できないが，一方で，アプリ本体のみを入力とする技術はソースコードの入手が困難な状況でも対策付与が可能である．

既存の改ざん対策技術においては，本来のアプリ動作に関する処理に加えて対策の為に追加で処理を行っている関係上，アプリの動作時間に対するオーバーヘッドが発生する．既存技術おいて発生するオーバーヘッドを比較すると，最も悪いケースでは，Protsenko らの手法においてアプリの処理時間に 500% のオーバヘッドがかかっていた．また，最も良いケースでは，SSN で約 7% のオーバーヘッドがかかっていた．なお，DashO については，公式サイトからオーバヘッドに関する情報を入手する事ができなかったが，前記の通り対策の原理上，一定のオーバーヘッドが生じると考えられる．

対策付与後のアプリの互換性については，Protsenko らの手法と Divilar においては対策によってアプリの互換性が失われ特定のバージョンの OS でしか対策付与後のアプリが動作しない問題がある．一方で，それ以外の手法では対策付与の後のアプリでも最新の OS 上で動作させる事が可能である．後方互換性が失われる問題は，改ざん対策を付与する際に用いる技術が，Dalvik 仮想マシンや，Android フレームワークなど OS に含まれるコンポーネントの実装に強く依存している場合に発生する．

なお，本節における既存技術の比較は，対策の堅牢性を考慮していない点については注意されたい．アプリの改ざん対策技術において，対策の強度を図る共通的な指標は現状では存在しておらず，既存研究においても定性的な評価に留まっている場合や，手法に特有の独自の評価指標を作った上で評価が行われている場合が多い．このような評価指標の問題は，アプリの改ざん防止技術に関する研究におけるオープンな問題の一つであると言える．

表 4.1　従来手法の比較

	入力データ	対策の効果	オーバーヘッド	後方互換性
Protsenko らの手法 [48]	アプリ	実行コードの暗号化，メモリダンプによる復号されたコード取得の妨害	〜500 %	なし
Divilar [58]	アプリ	オペコードのランダム化，既知ツールによる解析防止	9〜29.2 %	なし
SSN [40]	ソースコード	自己改ざん検知コードの隠蔽，デバッグの困難化	7〜12 %	あり
CrackProof [62]	アプリ	自己改ざん検知，アンチデバッグ，実行コード暗号化	〜10 %	あり
DashO [49]	ソースコード	難読化，自己改ざん検知	N/A	あり

4.4 脅威モデル

一般的に攻撃者が Android マルウェアを作成する方法には，フルスクラッチで一からマルウェアを作成する方法と，アプリの改ざんによって正規アプリに悪性な活動を行う機能を追加する方法の 2 つに大別される．攻撃者が後者のアプリ改ざんを行う背景には，マルウェアの検知回避を試みる際や，マルウェアを拡散させユーザ端末上に長期的感染させる上で，コストの面で優位性があるためであると考えられる．

マルウェアがフルスクラッチで作成されている場合，仮に当該マルウェアの一部機能が変更された亜種が出現した際にも，マルウェア全体に含まれるコードは元となったマルウェアとの間で類似している場合が多い．そのため，そのようなマルウェア亜種は既知のマルウェアの実行コードとの類似性に基づいた検知手法によって検知が可能である．仮に，攻撃者がそのような検知手法を回避する為にはマルウェアに含まれるコードの大部分を書き換える必要があり亜種の作成にコストが掛かる．一方で，アプリの改ざんによってマルウェアを作成する場合，改ざん元となる正規アプリの種類を変更する事で，マルウェア全体に含まれる実行コードのうち一定の割合の部分が大きく変化する．そのため，検査対象のマルウェア全体に含まれる実行コードと，既知のマルウェアの実行コードの類似性に基づいて検知を行うような手法を低コストで容易に回避できる．実際に，アプリ改ざんによって作成されるモバイルマルウェアの検知を目的とした様々な既存研究において，マルウェア全体に含まれる実行コードの中から改ざん元となったアプリに起因する部分と改ざんによって追加された悪性な部分を区別する事が課題となっている．

攻撃者がマルウェアを拡散させるためには，エンドユーザにマルウェアを正規のアプリと誤認させてインストールさせるための様々な方法が用いられる．典型的なマルウェア拡散の手段としては，サードパーティマーケット上でマルウェアを正規のアプリと偽って配信したり，SMS や SNS 等を経由したフィッシングによってユーザにマルウェアをインストールさせる方法などが挙げられる．攻撃者が人気の正規アプリの改ざんを行うのは，上記のような手法でマルウェアを拡散させエンドユーザによるインストールを促す上で，人気の正規アプリが有している人気や知名度を悪用する事が効果的であるためであると考えられる．加えて，マルウェアをエンドユーザの端末上に長期的に感染させ続けるという観点でもアプリ改ざんは効果的な攻撃手法である．ユーザがアプリをインストールする目的はアプリが提供する機能やコンテンツの利用である．そのため，マルウェアがユーザにとって意味のある機能やコンテンツを提供していない場合，仮にユーザ端末上にマルウェアを感染させる事に成功してもすぐにアンインストールされてしまい，長期的に感染させ続ける事が難しいと考えられる．これらを踏まえると，攻撃者が一からマルウェアを作成しユーザの端末上に長期的に感染させ続ける為には，ユーザがアプリを利用するモチベーションとなる機能を一から作成する必要がある．一方で，元のアプリが提供する機能を保持したまま悪性な機能を追加する改ざんを行う事で，攻撃者はユーザがアプリを利用し続けるモチベーションなる機能を作成するコストが発生しないメリットを享受できる．

4.5 改ざん対策技術に求められる要件

本節では 3 章における実験の結果，また 4.3 節，4.4 節で説明した既存技術と脅威モデルの内容を踏まえて，アプリの改ざん防止技術に求められる要件を整理する．

現状の正規アプリの多くは改ざん対策がなされておらず容易に改ざん可能である．攻撃者は，マーケット上から改ざん対策がなされていないアプリを入手した上で改ざんの標的にする事が可能であるため，理想的にはマーケット上で流通する全てのアプリ上に改ざん対策が実施されている事が望ましい．これを実現する為に

は，例えばアプリがマーケットに登録されるタイミングで改ざん検知機能を自動的に挿入する対策が考えられる．一方で，このような対策が適用できないケースも存在する． 4.3 節で述べた通り，既存の改ざん検知機能の付与技術では，対策の副作用としてアプリの動作にオーバーヘッドが生じる．よって動作の際にパフォーマンスが求められるアプリにおいてはマーケット登録時にこのような対策が自動的に付与されてしまうと，アプリが本来のパフォーマンスで動作せずユーザビリティ低下などの問題が発生する可能性がある．また一部の既存技術では，入力としてアプリのソースコードを必要とする．一方で，マーケット上での一括で対策付与を行う事を想定すると，アプリ本体のみを入力として改ざん対策を付与できる事が望ましい．

上記を踏まえ，本章では以下の要件を満たすアプリへの改ざん対策付与方式について検討する．

1. **対策の回避に対するロバスト性：**アプリに改ざん対策が付与されている場合，攻撃者はそれらの対策の無効化を試みる事が予想される．典型的な攻撃シナリオとして，攻撃者はアプリの実行コードの中から改ざん対策の実装箇所を抽出し，それらの実装自体を改ざんによって書き換える事で対策の無効化を行う．そこで本章では，対策の無効化に対する堅牢性を向上させるために，アプリ実行コードの中から対策の実装箇所が発見されにくくする方式を検討する．
2. **対策によって保護されるアプリの網羅性：**攻撃者は，マーケット上に存在する改ざん対策がなされていないアプリを容易に入手する事が可能である．これは現状のアプリ改ざん対策は開発者が自発的に行う改ざん対策に留まっており，開発者の知識や実装スキルに依存して対策が付与されない場合がある為である．よって，本章では世の中で流通するアプリを網羅的に保護する為に，アプリマーケット上で一括して改ざん対策を付与可能な方式を検討する．
3. **対象の非限定性：**対策手法によっては，対策を付与可能なアプリが限定されてしまう可能性がある．例えば，従来手法の中には対策の付与にソースコードを必要とするものがあるが，保護対象アプリの開発形態や，手法を適用する場面によっては，ソースコードが利用できない場合が存在する．本章では，アプリマーケット上での改ざん対策の付与を想定し，ソースコードを利用せずアプリ本体である APK ファイルのみを入力として改ざん対策を自動的に付与可能な方式を検討する．
4. **副作用の低減：**改ざん対策の付与後のアプリは，対策付与後も保護対象アプリが元々提供していた機能が正しく保持され，かつ対策によって発生するアプリ動作時のオーバーヘッドが十分に小さい事が望ましい．一方で，既存の改ざん対策付与技術においては，アプリの動作時間に対して一定量のオーバーヘッドが発生する．改ざん防止に関する処理が追加される原理上，オーバーヘッドをゼロにすることは難しいと考えられるが，アプリの提供する機能によっては発生するオーバーヘッドを最小限に抑えたいケースも考えられる．そこで本章では，保護対象のアプリの提供する機能に応じて開発者が対策によって発生するオーバーヘッドの量を調整する事が可能な方式を提案する．

4.6 提案方式

本節では 4.5 節にて説明を行った要件に基づいて，マーケット上でのアプリに対する改ざん検知機能を自動付与するシステムの構成方法について論じる．

要件 1 の対策の回避に対するロバスト性の観点では，既存技術である Luo らの提案手法 [40] による自己改ざん検知コードのランダムな挿入が有効であると考えられる．一方，Luo らの提案手法では，入力としてアプリのソースコードを必要とするため，要件 3 を満たさない．また，Luo らの手法は改ざん検知対策の付与によるアプリのオーバーヘッドを低減する為の処理が組み込まれているが，開発者が特定の機能におけるオーバー

ヘッドを低減したい場合などには対応できず，要件 4 を満たさない．上記を踏まえ，本研究では既存手法を拡張する事で， 4.5 節の全ての要件を満たすシステムの構成方法を提案する．

図 4.1 に，提案するシステムによる処理の流れを示す．まず，アプリ開発者はマーケット上に改ざん対策を付与する前のアプリとアプリ署名用の秘密鍵の登録を行う．また，アプリ開発者はアプリの中に動作パフォーマンスが求められる機能がある場合には，当該機能が実装されている実行コードの箇所を記述したアノテーションの情報をあわせて入力する．

アプリを受け取ったマーケット上では，入力に基づいて以下の２つの処理を行う．

1. **改ざん検知コード挿入位置の決定：**入力となるアプリのディスアセンブルを行い解析し，改ざん検知コードの挿入する候補となる箇所をバイトコード中から抽出する．この際，アプリの元の機能を実装したバイトコードの動作を阻害しない位置を改ざん検知コードの挿入位置として抽出する．アプリ開発者からアプリの動作にパフォーマンスが求められる機能の実装箇所に関するアノテーションの情報が登録されている場合には，当該箇所を検知コードの挿入候補点から除外する．最後に，上記の条件を全て満たす改ざん検知コードの挿入候補点からランダムに抽出した箇所を最終的な改ざん検知コードの挿入点とする．
2. **改ざん検知コードの挿入：**手順 1 で抽出した改ざん検知コードの挿入点に対して，改ざん検知を行う機能を実装したバイトコードを挿入する．この際，改ざん検知コードの実装として予め複数種類のテンプレートを用意しておき，ランダムに選択した改ざん検知コードのテンプレートが挿入される．
3. **アプリへの署名付与：**手順 2 で改ざん検知コードの挿入を行った後のバイトコードを再ビルドし，開発者によって登録された署名用の秘密鍵によってアプリに署名付与を行う

最終的に，上記の手順でアプリに対して改ざん検知コードを自動挿入した後に，アプリがマーケット上に公開される．

手順 1 における自己改ざん検知コードの挿入位置を決定する際には，Luo らの提案手法におけるアルゴリズムが利用可能である．Luo らの手法ではアプリの実行コードにおけるコントロールフローの解析結果に基づいて，元のアプリの機能を阻害しない改ざん検知コードの挿入位置を決定している．このようなコントロールフローの解析は，例えば Android におけるバイトコードの解析フレームワークである Soot フレームワーク [52] や Androguard [20] を用いる事でソースコードを入力とせずとも行うことが可能であるため，手順 1 の処理は既存手法の拡張によって実現可能である．また，手順 1 においては Luo らの手法に対して，開発者からのアノテーション情報に基づいて，自己改ざん検知コードの挿入候補点を限定している処理を追加している．これにより，開発者が特定の機能の実装においてオーバーヘッドを発生させたく無い場合当該箇所への自己改ざん検知コードの挿入を防止できる．

4.7 提案方式によるアプリ改ざん対策の効果

提案方式によりアプリマーケット上での自動的な改ざん対策の付与が可能となる．提案方式では，改ざん対策として自己改ざん検知コードを多数挿入する．自己改ざん検知コードの挿入位置，および自己改ざん検知コードの実装はランダム性を持って決定されるため，入力となるアプリごとに検知コードの挿入箇所や実装が異なる特性がある．そのため，攻撃者がアプリを改ざんする為には，ランダムな形態で複数箇所に実装された改ざん検知コードを全て無効化した上で目的の改ざんを行う必要がある．このような場合，攻撃者は個々のアプリの自己改ざん検知コードがどのような実装になっているかを解析する必要があるため，自己改ざん検知

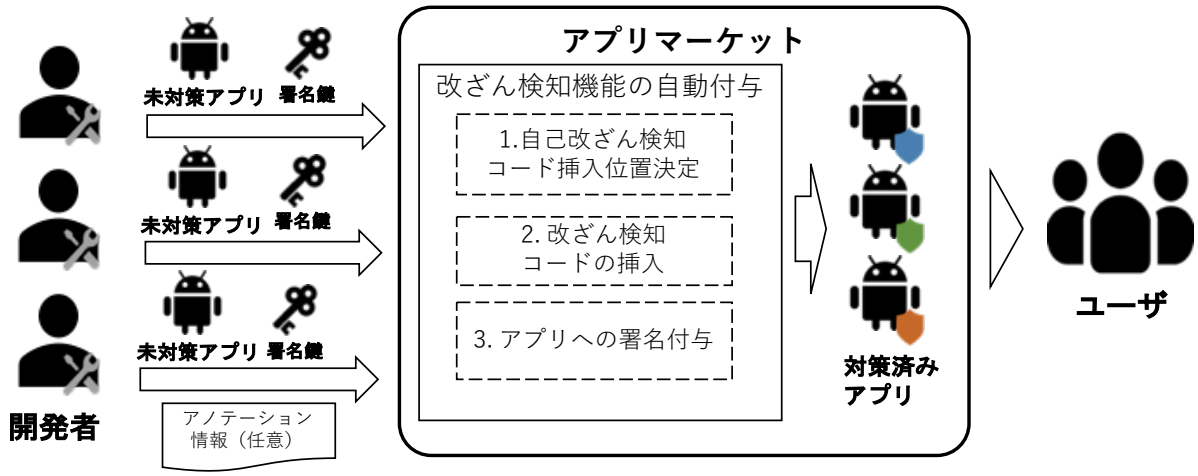

図 4.1　マーケット上でのアプリに対する改ざん検知機能の自動付与の流れ

コードの無効化および目的とするアプリ改ざんの全てを全自動で行う事が困難になる．

また，提案方式では，開発者がアノテーション情報を入力する事により対策によって発生するオーバーヘッドを調整する事が可能である．ただし，改ざん検知機能の挿入位置を限定する事は，攻撃者による解析の困難性を低下させる可能性がある．アプリの実行コード上で改ざん検知機能が挿入され得るスペース，つまり改ざん検知機能を挿入する際のランダム性の大きさは，発生するオーバーヘッドとのトレードオフの関係なると考えられるため，開発者は自身のアプリの資産性や改ざんによって受ける被害の大きさを加味した上で，任意の強度で改ざん対策を付与する事が可能である．このような実際の開発者が対策を利用する上での利便性の観点は既存研究では十分に検討されておらず，中でも自動的な改ざん対策の付与技術において発生するオーバーヘッドを開発者の視点から容易に調整可能にする技術はこれまでに提案されていなかった．

4.8　提案方式の拡張

提案方式では，アプリの堅牢化対策として自己改ざん検知機能の自動付与を行っているが，攻撃者による対策回避へのロバスト性という観点では，その他の堅牢化対策を組み合わせる事も有効であると考えられる．例えば，アプリに対して自己改ざん検知コードと合わせてアンチデバッグ機能を実装したコードも合わせて挿入する方式や，実行コードやリソースを暗号化するパッキングの処理をあわせて行う方式なども技術的に実現可能である．実際に，中国のあるサードパーティマーケットで，マーケットへのアプリ登録時に自動的にアプリのパッキングを行うサービスを提供している事例がある．同様に，アプリの動作において重要な機能をサーバ上に実装して，クライアント-サーバ形式でアプリを動作させる事も改ざん対策として効果的である．一方でこのような対策はクライアントとなる Android アプリ側を攻撃者が改ざん可能であるため，その他の対策との組み合わせが必須であると言える．上記のように， 4.5 節にて定義した要件を満たすアプリの堅牢化技術は，提案方式の拡張として利用可能であり，それらを組み合わせた多層的なアプリの堅牢化によりシステムの有効性を高める事が可能であると考えられる．

上記のようなアプリ堅牢化処理の拡張が考えられる一方で，対策によって発生する副作用には十分に考慮する必要がある．一般的に，アプリの堅牢化の為に実行コードやリソースが追加されると，処理時間へのオー

バーヘッドやアプリサイズの増加などの影響を避けられない．提案システムでは，処理時間に対するオーバーヘッドを開発者視点からある程度コントロール可能な仕組みを導入している．アプリの堅牢化処理の拡張を考える上でも同様に，対策による副作用を可能な限り小さくしたり，副作用の影響の強さを開発者の視点からコントロールできるようにする仕組みも同時に検討する必要がある．このような改ざん対策の強度とユーザビリティのトレードオフは，セキュリティ対策におけるオープンな課題であるといえる．そのため今後の手法の拡張にあたっては，実際の開発者を対象としたユーザスタディ等により手法のユーザビリティの定量/定性評価を交えた検討が重要であると言える．

4.9 まとめと今後の課題

本章では，アプリマーケット上で Android アプリに改ざん検知機能を自動的に付与する方式について提案した．提案方式によりマーケット上で流通するアプリが自動的な改ざんに対して堅牢となり，攻撃者がアプリ改ざんを用いて大量のマルウェアを作成する際のコストを増加させる事が可能である．提案方式では，既存技術の拡張により個々のアプリに対してランダム性を持った改ざん検知機能を多数挿入する．これにより，攻撃者は各アプリに特化した解析を実施し全ての改ざん検知機能を無効化した上で，目的の改ざんを行う必要があり，特に自動的な改ざんに対してアプリの堅牢性が向上する事が期待できる．また提案方式では，アプリ開発者がアプリ動作においてパフォーマンスを求める機能を実装している箇所をアノテーション情報として入力する事で，改ざん検知機能を挿入する場所を限定することにより，発生するオーバーヘッドを減少させ，対策がアプリの正常な動作に与える影響を小さくする事が可能である．ただし，改ざん検知機能の挿入位置を限定する事は，攻撃者による解析の困難性を低下させる可能性がある．アプリの実行コード上で改ざん検知機能が挿入され得るスペース，つまり改ざん検知機能を挿入する際のランダム性の大きさは，発生するオーバーヘッドとのトレードオフの関係なると考えられるため，開発者は自身のアプリの資産性や改ざんによって受ける被害の大きさを加味した上で，任意の強度で改ざん対策を付与する事が可能である．

今後の課題として，提案方式の有効性の定量的な評価が必要である．アプリの堅牢化技術が攻撃者によるアプリ改ざんに対してどの程度耐性を有しているかの定量的な評価は，現状では一般的な評価指標が存在しない事からオープンな課題であると言える．複数のアプリ堅牢化技術の強度を統一的な指標で比較する方法として，例えばアプリのリバースエンジニアリングに関する高度な知識を有した研究者に対策付与後のアプリに対して擬似的な改ざん攻撃を行ってもらい，攻撃にかかった時間や攻撃の成功率を計測するユーザスタディのアプローチが考えられる．同様に，実際に改ざん対策を付与した後のアプリをマーケット上で流通させ，改ざんによる被害をどの程度受けるかを計測する実証的なアプローチも考えられる．今後は，上記のようなアプローチによる提案手法の有効性評価について継続的に検討していきたい．

提案方式は攻撃者が改ざんにかかるコストを増加させる上で有効であると考えられるが，攻撃によって得られる利益が十分に大きい場合には，攻撃者はコストを払って個々のアプリに特化した解析を行いアプリ改ざんを行う場合も考えられる．よって，攻撃の抑制を目的とした場合，攻撃自体にかかるコストを増加させる対策と合わせて，攻撃によって得られる利益を減少させる対策も実施する事が効果的である．これを踏まえ，5 章では攻撃者がモバイルアプリを標的とした攻撃によってマネタイズを行う主な手段の一つであるオンライン広告不正に着目し，不正なマネタイズを防止するための検知手法について提案する．

第 5 章

オンライン広告不正の検知手法の提案および実態調査

5.1 はじめに

オンライン広告はモバイルアプリや Web サイトにおける主要なマネタイズ手段であり，近年市場規模が急速に拡大している．Interactive Advertising Bureau（IAB）による報告 [6] によれば，2018 年における英国でのオンライン広告の収益は 107.5 億ドルに登るとされている．このような広告市場の拡大に伴い，広告収入を目的としたソフトウェア (アドウェア) や，自動化された広告クリック (ボット) によって広告収益を詐取する広告不正の被害も拡大している．IAB による調査 [8] によれば，米国では広告不正により毎年 8.2 億ドルの損害が発生しているとされている．

これまでに，オンライン広告によって発生する通信の分析や，広告を表示するクライアントアプリケーション（例：モバイルアプリケーション，ブラウザ拡張機能 など）の分析により，広告不正のメカニズムを調査した様々な研究が行われている．また，異常検知によるクリックパムの検知 [18, 19] や，クラウドソーシングを悪用した広告不正の検知 [51] など，広告不正の検知手法も合わせて研究されている．

昨今の広告不正にはさまざまなタイプが存在する．中でも特に検知が難しい高度な攻撃として，多数のクライアント，パブリッシャを悪用した分散型の広告不正が挙げられる．分散型の広告不正においては，攻撃者によってコントロールされている多数の悪性クライアントから発生する広告通信が，正規ユーザによって発生する広告通信に混じってしまう．このような攻撃は，例えば広告通信のバースト性のようなシンプルな特徴量を用いた異常検知ベースの検知手法では検知が難しく，大量の見逃しや誤検知が発生する．

本研究では，広告ネットワーク上で観測された不正な広告リクエストを高精度に検知する手法を提案する．提案手法は機械学習に基づく手法であり，広告リクエストから抽出した特徴量を利用して，それらを悪性/良性のどちらかに分類する．本研究では提案手法の設計において，攻撃者による検知の回避に対してロバストな特徴量を設計する為に “攻撃者は広告ネットワークの立場から広告リクエストを観測する事ができず，正規ユーザから発生する広告リクエストの統計を知ることができない” という仮定を導入した．提案手法は上記の仮定に基づいて，クライアントおよびパブリッシャーに関する情報から統計的な特徴量を算出し利用することで，正規ユーザによる大量の広告リクエストの中から広告不正を検出する．

以下は，本章で説明する研究における貢献である．

- 広告ネットワーク上で観測されるリクエストの中から広告不正に関連するリクエストを高精度に検出可

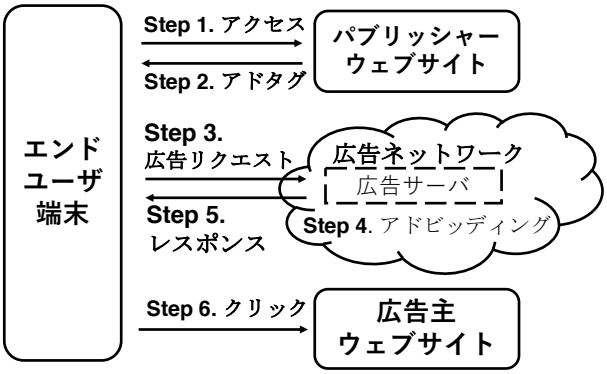

図 5.1　オンライン広告が表示される流れ

能な手法を提案した．提案手法は，攻撃者の立場からコントロールが難しい特徴量を用いており対策の回避に対してロバストである．

- 評価の結果から，提案手法は広告不正を効率的に検知可能である事を確認した．具体的には，提案手法において導入した特徴量により，不正なリクエストのバースト性に基づいた従来の検知手法と比較して，Recall 値が 10% 向上する事を確認した．
- 長期間のデータセットを用いた評価の結果，提案手法は従来手法と比較して時間経過による検知率の低下が起こりにくい事を確認した．
- 提案手法を実際の広告ネットワーク上で観測/収集された大規模なログに対して適用することで，現状の広告不正に関する実態調査を行った．調査の結果から，800 万件の広告閲覧のうち約 6.9% が広告不正に起因している事を確認した．
- 実態調査を通じて，最新の広告不正に関する複数の特性を明らかにした．例えば，大量の広告不正がクラウドサービスによって管理されている IP アドレスから発生している事例や，攻撃者が広告不正を行う際に特定の OS バージョンを利用してる（もしくは利用していると見せかけている）事例などを確認した．

5.2　オンライン広告と脅威

本節では，背景としてオンライン広告のメカニズムや脅威モデルについて説明する．

5.2.1　オンライン広告のメカニズム

オンライン広告における主なステークホルダーは，広告を閲覧する**エンドユーザ**，広告表示枠を提供する**パブリッシャ**，広告を出稿する**広告主**，パブリッシャと広告主の間を仲介する**広告ネットワーク**，の 4 種類である．

図 5.1 に，ユーザがブラウザ等によって Web サイト閲覧した際に表示される広告（以降，Web 広告）の仕組みについて説明する．まず，エンドユーザは広告を含む Web サイトにアクセスする（Step 1）．その際，パブリッシャが自身の Web サイト上に埋め込んだアドタグ（例えば，HTML における`<script>`タグや`<iframe>`タグ）がエンドユーザのブラウザ上に読み込まれる（Step 2）．次に，アドタグにより実行される JavaScript か

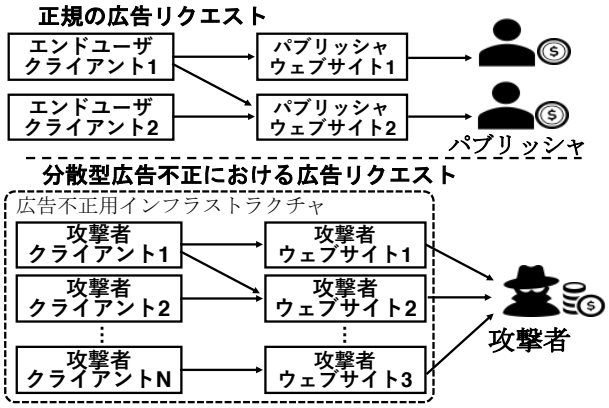

図 5.2　分散型広告不正の流れ

ら広告ネットワーク事業者が管理するサーバに対して，広告を要求するリクエストが送信される（Step 3）．この時，アドネットワーク上でアドビッディングが行われる（Step 4）．アドビッディングでは，アドタグによって送信されたパブリッシャ Web サイトの情報やクライアント環境の情報（例えば，閲覧中のページカテゴリ，広告枠のサイズ，OS バージョン，Cookie など）を元に広告枠の競売が行われる．最終的に入札を行った広告主の広告がリクエスト元の広告枠に表示される（Step 5）．また，ユーザは，表示された広告をクリックすることで，広告主の Web サイトへリダイレクトされる（Step 6）．広告が表示される際には，バナー画像などが読み込まれる他に，広告効果の測定を目的とした情報収集用のスクリプトが同時に読み込まれ，それらによりユーザの広告閲覧や広告クリックが記録される．

オンライン広告において実際に支払われる広告費は様々な指標によって決定される．典型的には，ユーザに広告が 1000 回表示されるごとに広告費を支払う Cost Per Mille（CPM）や，広告が 1 回クリックごとに広告費を支払う Cost Per Click（CPC），広告を経由したユーザが，商品の購入などの広告主が目的とするアクションを行なうごとに広告費を支払う Cost Per Action（CPA）どの指標が用いられる．

5.2.2　Web 広告とアプリ内広告

ブラウザ上に読み込まれた Web コンテンツにより表示される Web 広告の他に，モバイルアプリ内に組み込まれた広告 SDK により表示される広告 (以降，アプリ内広告) も存在する. ア プリ内広告は，有料アプリや課金コンテンツと同様に，モバイ ルアプリにおける主要な収益源である. アプリ開発者は，広告 ネットワーク事業者から提供される広告 SDK をアプリに組み 込むことで，自身のアプリ内に広告表示枠を用意する. ユーザがアプリを利用する際に，例えば画面下部に表示されるバナー 広告や，画面切り替わりのタイミングで表示されるインタースティシャル広告の形態でアプリ内広告が表示される. また，こ れらのアプリ内広告の表示には，アプリの中で Web コンテン ツを表示するためのコンポーネントである，WebView [7] の機 能を用いて実装されている場合が多い. 典型的には，WebView 内に読み込まれた JavaScript コードにより，広告要求のリクエ ストの送信や，広告コンテンツの読込が行われる．

5.2.3 オンライン広告における脅威モデル

広告不正とは，実際には広告効果の発生しない不正な手段を用いて広告収益を詐取する攻撃である．オンライン広告はモバイルアプリの利用者や Web サイトにアクセスしたユーザなど，人間によって閲覧される事で広告効果を発生させる事を目的としている．一方で，攻撃者は広告効果が発生しない不正な手段を用いて広告閲覧をおこなう．このような攻撃はインプレッション詐欺に分類され，大量の無意味な広告閲覧を発生させる．同様に，攻撃者はエンドユーザが意図しない広告クリックを機械的に発生させる攻撃も行う．このような攻撃はクリック詐欺に分類される．

攻撃者は，様々な手段を用いて広告閲覧や広告クリックの水増しを行う．攻撃者の戦略は，大まかに人間による攻撃とプログラムによる攻撃の 2 つのグループに分類される．前者の戦略では，攻撃者によって雇われた複数の人間により，手動で攻撃が行われる．例えば，攻撃者はクラウドソーシングサービスを用いて低コストで労働者を雇い，特定のモバイルアプリや Web サイトから大量の広告リクエストを発生させる [51]．また，モバイルアプリや Web サイト上でユーザを騙して広告クリックを誘導する攻撃のような，パブリッシャの立場からの広告不正も存在する [50]．後者の戦略では，攻撃者はプログラムによって自動化された広告リクエストを発生させる．例えば，攻撃者は自動クリックプログラムを自身の PC 上で実行し攻撃を行う．同様に，マルウェアやアドウェア，PUP（Potentially Unwanted Programs）に感染したマシンを悪用して，自動的な広告リクエストを発生させるケースも存在する [17, 47]．

上記のような攻撃に加えて，より高度な技術を有した攻撃者は分散型の広告不正攻撃により対策の回避を試みる．図 5.2 は，分散型の広告不正攻撃の流れを示している．まず攻撃者は，多数の自身の管理下にある多数のクライアント/Web サイトで構成される，攻撃用のインフラストラクチャを用意する．その後，インフラストラクチャを構成する多数のクライアントから Web サイトに対して限れられた頻度でアクセスを行う．これらのリクエストは，大量のマシン（例：ボットネット）から送信される場合もあれば，少数のマシンから様々なクライアント環境を模して送信される場合（例：Cookie やユーザエージェントの偽装，HTTP プロキシの利用 など）もある．このようなタイプの攻撃は，攻撃者によって管理される各クライアントからの広告リクエストの量が正規ユーザによる広告リクエストの量と変わらないため，従来手法におけるリクエストのバースト性を用いた手法では検知が困難である．

表 5.1　提案手法で利用する特徴量の一覧（従来特徴量は既存研究 [50] に基づいて設計）

	No.	特徴量名	説明	特徴量の形式	特徴量の抽出元
新規設計された	1	rDNS-e2LD_freq	クライアント IP アドレスの逆引き結果から抽出された e2LD の出現頻度	数値	クライアント
特徴量	2	FQDN_freq	パブリッシャーの URL から抽出された FQDN の出現頻度	数値	パブリッシャ
	3	rDNS-e2LD and FQDN	rDNS-e2LD と FQDN の組み合わせ出現頻度	数値	クライアント & パブリッシャ
	4	geoip	クライアント IP アドレスの地理情報	カテゴリ変数	クライアント
	5	dynamic_range	クライアント IP アドレスが動的 IP アドレスレンジに存在するかどうか	2 値変数	クライアント
	6	alexa_rank	パブリッシャ URL における Alexa Rank の値	数値	パブリッシャ
従来手法における	7	impression_count	クライアントの広告閲覧数	数値	クライアント
特徴量	8	ctr	クライアントの広告クリック率	数値	クライアント

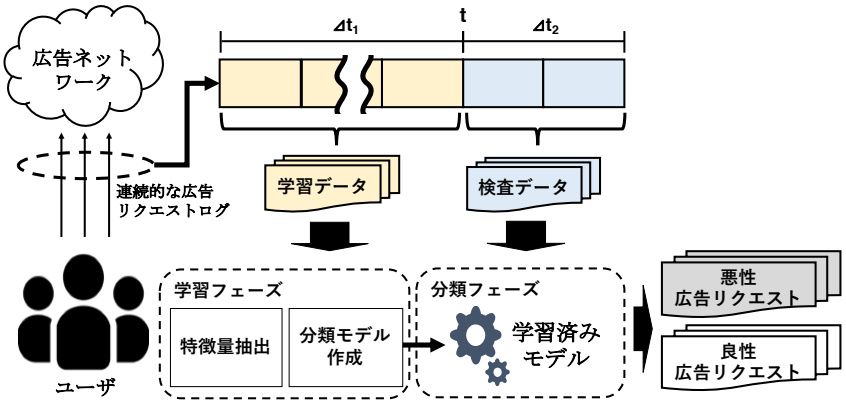

図 5.3　提案手法の流れ

5.3　提案手法

本節では，広告不正の検知を目的とした提案手法について説明する．静的なルールベースの検知（例：広告リクエストパラメータにおける不備の抽出）は，検知ルールを作成する上で手動での手間が発生するためスケールしない問題がある．そのため，昨今の研究では広告リクエストのバースト性に基づいた異常検知による手法が研究されている．一方， 5.2 節で述べたとおり，攻撃者は分散型の広告不正により発生する広告リクエストの量を容易にコントロール可能であるため，広告リクエストの量を検知における特徴量として用いた手法では，見逃しや誤検知が発生する問題がある．

上記を踏まえ，本研究では広告不正を高精度に検知する手法を提案する．本研究では，以下の仮定を導入し提案手法の設計を行った：“攻撃者は広告ネットワークに到達する大量の広告リクエストを観測する事ができないため，正規ユーザからの広告リクエストにおけるクライアント環境やパブリッシャ URL に関する統計を知り得ない”．具体的には，クライアント環境，パブリッシャ URL，およびそれらの組み合わせの出現頻度を広告ネットワーク上で観測された広告リクエストのログから抽出し特徴量として用いた．攻撃者は観測点の違いから上記の情報を知りえないため，攻撃者の管理下にあるクライアントおよびパブリッシャ Web サイトに起因する広告リクエストは，正規ユーザからの広告リクエストと異なる傾向を示す．提案手法では，上記の統計値を含む特徴量ベクトルを作成した上で，機械学習により広告リクエストを良性/悪性のどちらであるかを分類するモデルを生成する．なお，提案手法は広告ネットワークと連携の上広告不正を検知する事を想定し設計されている．そのため，広告ネットワーク内に正規ユーザからの大量の広告リクエストが観測可能な悪性なステークホルダーが存在するような攻撃については，検知の対象外とする．以下の節では，提案手法の詳細について説明する．

5.3.1　提案手法の概要

図 5.3 に提案手法による処理の流れを示す．提案手法は学習フェーズと分類フェーズの 2 つのフェーズによって構成される．学習フェーズにおける入力は，正規もしくは悪性の正解ラベルを含む連続的な広告リクエストのログである．ある時点 t を定義した際に，Δt_1 から t までの期間に広告ネットワーク上で観測された広

告リクエストを学習用データセットとして用いる．学習フェーズでは，学習用データセットに含まれる広告リクエストログから特徴量を抽出し機械学習による分類モデルを作成する．分類フェーズにおける入力は，学習フェーズで作成された分類モデルおよび t から Δt_2 の期間に観測された連続的な広告リクエストのログである．提案手法の最終的な出力は，分類フェーズで入力とした各広告リクエストログが悪性/良性のどちらであるかを検査した結果である．

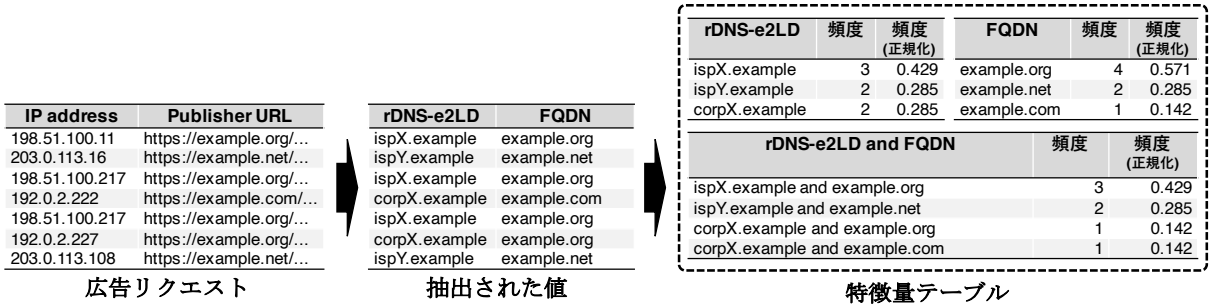

図 5.4　提案手法による特徴量算出の流れ

5.3.2　特徴量の抽出

提案手法では，広告ネットワーク上で観測された広告リクエストから 8 つの特徴量を抽出する．表 5.1 に，提案手法で用いられた特徴量の一覧を示す．No.1–6 の特徴量は本研究において新たに設計された特徴量であり，No.7, 8 の特徴量は既存研究 [50] において提案されている特徴量である．以下，各特徴量の詳細について説明する．

新たに設計された特徴量: 表 5.1 ににおける No.1–3 の特徴量は，広告ネットワーク上で観測される全ての広告リクエストにおけるクライアント環境およびパブリッシャ Web サイトの出現頻度に基づいて算出される特徴量である．図 5.4 に，特徴量を算出する際の流れを示す．まず，学習用データセットから以下の 2 つ値を抽出した上で，それらにおけるユニークな値が何回出現したかをカウントする．

- **rDNS-e2LD_freq:**
  提案手法では，クライアント IP アドレスの DNS 逆引き結果から抽出した effective second-level domain（e2LD）の出現頻度を計算する．e2LD は，インターネット上でユーザが登録可能な最小単位のドメイン名のユニットである．例えば，あるドメイン名 `foo1.example.co.jp` が存在した場合，その e2LD は `example.co.jp` となる．任意のドメイン名における e2LD は，Public Suffix List [4] を利用することで抽出可能である．一般的に，ある組織において特定の目的で利用されている IP アドレス（例：インターネットサービスプロバイダー（ISP）による動的 IP アドレス，企業における社内ネットワークで利用される IP アドレス など）は，DNS 逆引き結果における e2LD（rDNS-e2LD）が共通である特徴がある．なお，提案手法では，IP アドレスそのものではなく，rDNS-e2LD をクライアント環境を代表する特徴量として用いる．なぜならば，一般的に rDNS-e2LD は IP アドレスそのものより攻撃者による変更が難しく，より安定した特徴量であると考えられるためである．
- **FQDN_freq:** 提案手法では，パブリッシャの URL から抽出した fully qualified domain name (FQDN) の出現頻度を計算する．なお，提案手法では URL そのものではなく，FQDN をパブリッシャを代表す

る特徴量として用いる．なぜならば，攻撃者にとってパブリッシャサイトの為に新たな FQDN を用意する事は，新たな URL を用意するよりコストがかかるためである．

提案手法では，上記の 2 つの特徴量と合わせて，それらの組み合わせ出現頻度を計算する（**rDNS-e2LD and FQDN**）．組み合わせ出現頻度の計算では，学習用データセットにおける rDNS-e2LD と FQDN における全てのユニークな値の一覧を取得した上で，各値の組み合わせが出現した回数をカウントする．図 5.4 における右側の表は，算出された各値の出現頻度を示している．最終的に，出現頻度の値は，0 から 1 の間に正規化され特徴量として利用される．検査対象のデータセットから特徴量を抽出する際，学習用データセット内に共通する rDNS-e2LD や FQDN が存在した場合，それらの出現頻度を特徴量として利用する．また，共通する rDNS-e2LD や FQDN が存在しなかった場合，特徴量の値は 0 とする．

例えば，図 5.4 における左側のような学習用データセットが与えられ，検査対象の広告リクエストにおける rDNS-e2LD，FQDN がそれぞれ，*ispX.example* と *www.example1.jp* である場合，特徴量は以下の通り決定される：*rDNS-e2LD = 0.429, FQDN = 0, rDNS-e2LD and FQDN = 0* .

上述の特徴量と合わせて，提案手法では以下の 3 つの特徴量を用いる．なお，カテゴリ特徴量（geoip）は，One-hot-encoding によってベクトル化する．

- **geoip:** 提案手法では，GeoIP [2] を用いて決定されたクライアント IP アドレスの国情報を特徴量として用いる．攻撃者が広告不正を行う際に，攻撃元として分散した IP アドレスを用いる事が予想されるが，一方で，一般的に多数の国における IP アドレスを大量に用意するにはコストがかかる．本特徴量は，攻撃に用いられる IP アドレスが特定の国に偏るという仮定に基づいて設計された．
- **dynamic_range:** 提案手法では，クライアント IP アドレスが，ISP による動的 IP アドレスのレンジに含まれているかどうかを特徴量として用いる．既存研究 [14, 35, 55] と同様に，提案手法ではクライアント IP アドレスの逆引き結果に対してキーワードマッチング (例：*dhcp*, *ppp* など) を行うことで，IP アドレスが動的 IP アドレスアドレスのレンジに含まれているかどうかを判断した．本特徴量は，ネットワークレベルの特性は攻撃者にとって容易に変更できない，という仮定に基づいて設計された．
- **alexa_rank:** 提案手法では，パブリッシャ URL の知名度を Alexa Topsites List [1] によって算出した値を特徴量として用いる．具体的には，パブリッシャ URL から抽出した FQDN を Alexa のトップ 100 万のリストとマッチングさせた．もし FQDN がリストに含まれていた場合，その順位を特徴量として用いた．また，FQDN がリストに含まれていなかった場合，100 万 +1 を特徴量として用いた．本特徴量は，広告不正に用いられる Web サイトは正規のユーザを誘導するようなコンテンツが少なく知名度が低いという仮定に基づいて設計された．なぜならば，そのような Web サイトを構築・管理することは攻撃者にとって高コストであり，それらの知名度は低くなる傾向があるためである．

Conventional Features: 提案手法では，既存研究 [50] で提案されている広告リクエストのバースト性に基づいた 2 つの特徴量を用いる．このような指標は攻撃者にとって容易に変更可能である一方で，高度でない単純な広告不正を検知する上では効果的である．

- **impression_count:** 提案手法では，各クライアントにおける広告閲覧の回数を特徴量として用いる．具体的には，学習用データセットにおける各クライアントが広告を閲覧した回数をカウントし正規化した値を特徴量とする．検査用データセットにおいて特徴量を算出する際には，クライアントが学習用データセットと検査用データセットに共通して存在する場合には，学習用データセットから算出された値を用いて，そうでない場合には特徴量の値を 0 とする．

表 5.2　評価に用いたデータセット

データセット名	インプレッションログの件数	クリックログの件数	データ収集日時（期間）	正解ラベルの有無
データセット A	13,814,768	37,970	2017 年 10 月 29 日 (24 時間)	部分的に正解ラベル有り (1%)
データセット B1	208,645	708	2018 年 6 月 6 日 (1 時間)	正解ラベルあり
データセット B2	242 411	893	2018 年 6 月 13 日 (1 時間)	正解ラベルあり
データセット B3	253,083	978	2018 年 6 月 20 日 (1 時間)	正解ラベルあり
データセット B4	323,074	954	2018 年 6 月 27 日 (1 時間)	正解ラベルあり

- **ctr:** 提案手法では，各クライアントにおける広告閲覧数に対するクリック数（以下，クリック率）を特徴量として用いる．具体的には，学習用データセットにおける各クライアントのクリック率の値を特徴量として用いて，検査用データセットにおいて特徴量を算出する際には，*impression_count* と同様の方法を用いて特徴量を算出する．

上記の特徴量を算出するためには，データセットにおける各クライアントおよび各クライアントからの広告リクエストを区別する必要がある．本研究では，広告リクエストにおける IP アドレスとユーザエージェントのユニーク値の組み合わせにより，各クライアントの区別を行う．本アプローチによる制約については， 5.6 節にて議論する．

5.3.3　学習および分類手法

提案手法では，正解ラベル付きの学習用データセットから機械学習による分類モデルを作成する．機械学習のアルゴリズムには，大量のデータセットを並列に処理可能な Random Forest [12] を用いる．Random Forest は，学習済みの分類モデルから，各特徴量がどの程度分類に寄与したかの重要度スコアを取得する事が可能であり，モデルの解釈性の観点から他のアルゴリズムより優れている．Random Forest における各種パラメータは実験的に決定される．具体的には，事前に少数のデータセットでパイロット実験を実施した上で，最も分類精度が高くなったパラメータの値を本実験に用いる．また，学習用データセットにおいて，悪性/良性のラベルに偏りが存在する場合，各ラベルの数が等しくなるように，重み付きのランダムサンプリングを用いる．

5.4　提案手法の評価

本節では，提案手法の検知精度に関する評価の結果について述べる．評価において，既存手法で用いられている広告リクエストのバースト性に基づいた特徴量と，提案手法において新たに設計された特徴量の比較を行った．また，長期間のデータセットを用いることで，時間経過による検知精度の低下についても合わせて評価を行った．最後に，ある広告ネットワーク上で 24 時間の間に観測された広告リクエストの大規模ログに対して提案手法を適用することで，昨今の広告不正の実態調査を行った．なお，提案手法は，Python と scikit-learn [5] パッケージを用いて実装された．また，クライアントの OS バージョンをユーザエージェントから推定する際には ua-parser [7] を用いた．以降の全ての実験は，16-core 2.8GHz Xeon CPU と 128-GB のメモリを搭載した Ubuntu サーバ上で行われた．

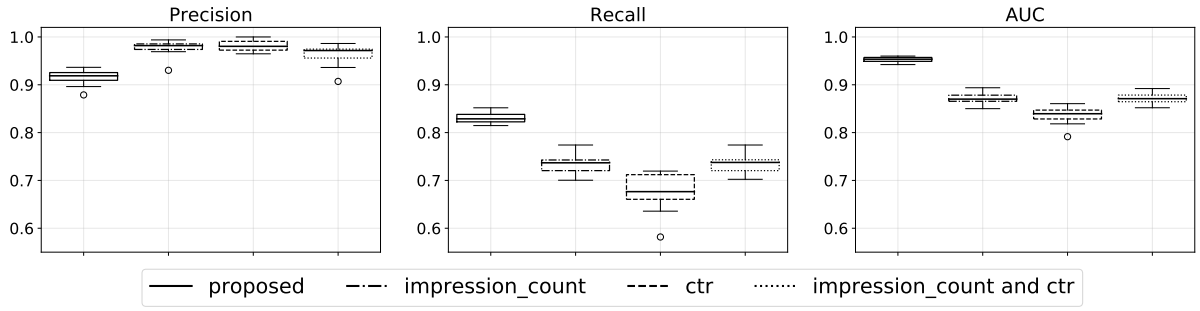

図 5.5　各分類モデルにおける precision, recall および AUC

5.4.1　データセット

表 5.2 に本評価で用いたデータセットの内容を示す．データセット A には，1 日の間に観測された全ての広告リクエストログが含まれる．本データセットは，提案手法の検知精度の評価，および広告不正の実態調査に用いられた．データセット B1–B4 は，時間経過による検知精度の低下を評価する際に用いられた．なお，データセット B1–B4 はとある 1 日に観測された全ての広告リクエストログからある 1 時間のログをサンプリングしたものである点に注意されたい．広告リクエストのログには，インプレッションログ，およびクリックログが含まれている．１件のインプレッションログは，1 つの広告がユーザの利用するモバイルアプリやブラウザ上に表示された事を表すログである．同様に，１件のクリックログは，ある広告がユーザによって１回クリックされた事を表すログである．これらのログは，とある日本の広告ネットワーク上で広告不正対策サービスを行っている企業 [3] によって収集された．インプレッションログには，広告が閲覧された時刻のタイムスタンプ，広告が表示されたパブリッシャ Web サイトの URL（パブリッシャ URL），広告を閲覧したユーザの IP アドレス（クライアント IP アドレス），広告の閲覧に用いられたブラウザのユーザエージェントなどの情報が含まれている．クリックログには，広告がクリックされた際の X-Y 座標や，広告クリックが発生した時刻のタイムスタンプの情報などが含まれている．また，上記の二種類のログには，広告閲覧と広告クリックの対応，つまりユーザがどの広告をクリックしたのかを紐つける為のインプレッション ID が含まれている．以降の評価においては，インプレッション ID によってインプレッションログとクリックログを結合した状態で分析を行った．

本評価においては，手動での正解ラベル付与を行った．教師あり機械学習のアプローチにおいて，正解ラベルは不可欠なものである．一方で，既存研究 [18] で言及されている通り，広告不正検知において広告リクエストに関する良性/悪性の正解ラベルを得ることは困難である．例えば，ある広告クリックが広告不正によるものであるかを厳密に判断するためには，ユーザに対して広告を意図的にクリックしたか否かを尋ねる必要がある．このような正解ラベル付与のアプローチは実用的でなく不可能に近い．そのため本研究では，データ提供元の広告不正対策サービスにおけるオペレータによって手動で解析された少数のログを正解ラベル付きデータセットとして用いた．オペレータによる具体的な解析の流れやアルゴリズムは非公開であるが，当該企業の公式 Web サイトにおける記載によれば，広告不正に特有な複数の指標（例：ブラウザ情報，IP アドレス，広告閲覧者の挙動，既知の攻撃グループの情報 など）を組み合わせた検知が行われている．正解ラベルの生成方法に関する制約事項については，5.6.1 節にて詳細を述べる．

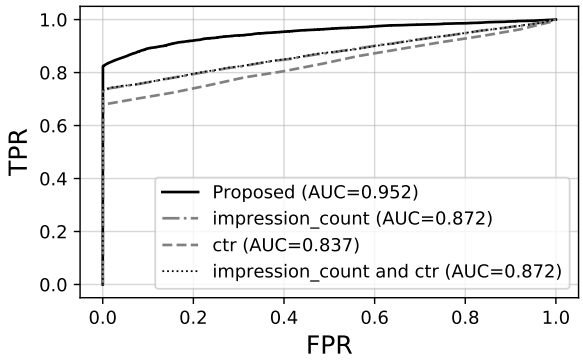

図 5.6　分類結果における ROC 曲線

5.4.2　提案手法の効率性

少数のデータセットを用いて提案手法の検知精度について評価を行った．まず，データセット A から 1% のログ（138,136 件）をランダムにサンプリングした後に，手動での正解ラベル付与を行った．次に，それらを複数の学習用/検査用データセットのペアに分割した．なお時間の間隔に関するパラメータ Δt_1 と Δt_2 をそれぞれ 12 時間と 1 時間とした．また，起点となる時間 t を 12:00 から 23:00 まで 1 時間ごとずらしてデータセットのペアを作成した．まとめると，本評価は 12 組の学習用/検査用データセットによって実施された．例えば，最初の学習用データセットは 00:00 から 12:00 の間に収集され，対応する検査用データセットは 12:00 から 13:00 の間に収集された．

本研究では，precision, recall, area under the ROC curve (AUC) の値を評価基準として用いた．なお，precision と recall は以下のように定義される．

$$precision = \frac{\text{悪性であると正しく検知された広告リクエストの数}}{\text{検知された全ての広告リクエストの数}}$$

$$recall = \frac{\text{悪性であると正しく検知された広告リクエストの数}}{\text{検査用データセットに含まれる全ての悪性な広告リクエストの数}}$$

広告不正検知手法において，precision の値が高くなればなるほど，誤検知が減りパブリッシャーの利益を守る事ができる．同様に，recall の値が高くなればなるほど，見逃しが減り広告主の利益を守る事ができる．広告ネットワークの視点では，precision や recall の値は利益の観点からは直接的な影響を受けないが，検知の精度によっては，広告ネットワーク自体の評判に対して風評的な被害を受ける可能性がある．

提案手法において新規に設計された特徴量の有効性を評価する為に，提案手法による分類モデルと合わせて，既存研究で提案されている広告リクエストのバースト性に基づいた特徴量である *impression_count*， *ctr*，および *impression_count* と *ctr* の両方のみを用いた分類モデルを合わせて作成し検知精度の比較を行った．なお，既存研究におけるいくつかの検知手法では，広告リクエストのバースト性以外の特徴量が用いられている．一方で，それらの手法では広告リクエストに含まれるより多くの情報を用いているため適用可能なデータが限られる．今回の我々のデータセットには，それらの手法で用いられている情報（例：広告主を区別する為の ID，広告の単価の情報 など）が含まれていないため，直接的な検知精度の比較が困難である．そのため，本研究では最も基礎的な検知手法であるバースト性に基づいた既存手法をベースラインとして比較評価を行

表 5.3　提案手法において用いられた特徴量の重要度スコア（上位 20 位）

特徴量名	重要度スコア	特徴量の区分
impression_count	0.27931	既存研究による特徴量
rDNS-e2LD_freq	0.19556	新規に設計された特徴量
ctr	0.14630	既存研究による特徴量
geoip_JP	0.12422	新規に設計された特徴量
rDNS-e2LD and FQDN	0.07677	同上
FQDN_freq	0.05661	同上
geoip_US	0.05029	同上
geoip_None	0.03566	同上
alexa_rank	0.01953	同上
is_dynamic_range	0.01221	同上
geoip_TW	0.00278	同上
geoip_CN	0.00014	同上
geoip_MY	0.00009	同上
geoip_TH	0.00008	同上
geoip_CA	0.00007	同上
geoip_KR	0.00006	同上
geoip_HK	0.00006	同上
geoip_PH	0.00005	同上
geoip_VN	0.00004	同上
geoip_SG	0.00004	同上

う．その他の特徴量を用いた既存手法との比較は今後の課題とする．

図 5.5 に，各分類モデルにおける precision，recall, AUC の値を箱ひげ図で示す．提案手法による分類モデルは，従来手法による分類モデルと比較して precision の値がわずかに低いが，全てのデータセットのペアにおいて recall の値が既存手法より優れている．precision および recall の値の平均は，提案手法の分類モデルにおいてはそれぞれ 0.91, 0.83 であり，もっとも検知精度が高かった既存手法の分類モデル（impression_count のみを利用）では，それぞれ 0.97, 0.73 であった．この結果は，我々が新たに設計した特徴量が recall の値の改善に寄与しており，広告不正の見逃し低減に効果的であることを示している．既存手法と比較して提案手法ではおよそ 650–1,000 件の見逃しが低減された．本評価では，検査用データセット全体で 86,523 件の広告リクエストログが含まれており，また，1 日における全ての広告リクエストログで構成されているデータセット A には，12,815,768 件の広告リクエストのログが含まれていた．よって本結果は，提案手法をデータセット A 全体に適用した場合に，既存手法と比較して一日辺り約 100,000–160,000 件の見逃しを低減可能である事を示している．図 5.6 は，各手法における ROC 曲線を示している．なお，各 ROC 曲線は各データセットペアによる結果を結合したものである点に注意されたい．提案手法は，既存手法と比較して約 8.1 – 11.6% AUC の値が高くより効果的な手法である事が分かる．

表 5.3 は，提案手法によって用いられた各特徴量における重要度スコアの平均値（上位 20 件）を示してい

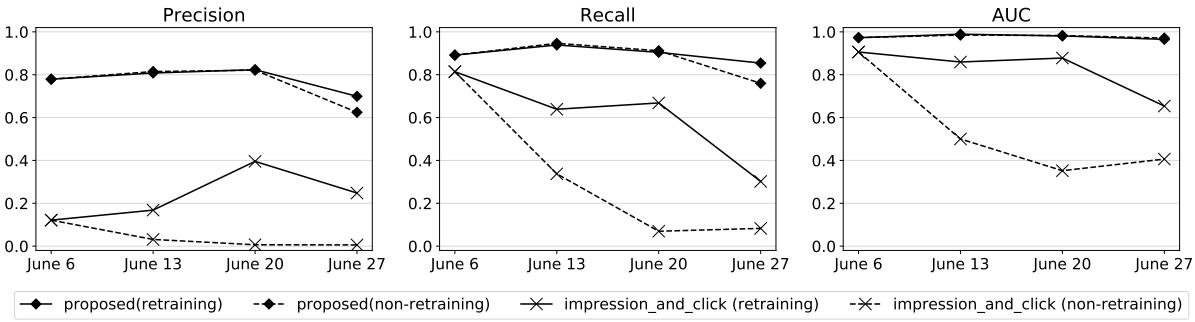

図 5.7　時間経過による precision, recall および AUC の低下

る．本研究において新たに設計された，rDNS-e2LD, FQDN およびそれらの組み合わせの出現頻度に基づいた特徴量は上位 6 位に含まれている事が分かる．同様に，既存研究において提案されている特徴量の重要度が，全体の重要度のうち 43% を占める事が分かる．これは，広告リクエストのバースト性に基づいたシンプルな特徴量も依然として広告不正の検知に有効である事を示している．

各分類モデルにおいて，学習および分類にかかる計算時間の比較を行うと，提案手法と既存手法の間で大きな差は見られなかった．提案手法においては，約 50,000 件のログから特徴量を抽出する際に，5–10 秒程度（1 件のログ辺りに 0.0001–0.0002 秒）の時間がかかっているが，これらの時間は十分に小さく無視できる影響である．

5.4.3　時間経過による検知精度劣化

長期間のデータセットに対して提案手法，および既存手法を適用する事で，時間経過による検知精度の低下について評価を行った．表 5.2 におけるデータセット B1–B4 が本評価に用いられた．なお，これらのデータセットはある月における複数の日付で観測された広告リクエストのログで構成されている．本評価では，以下の二種類の方法で評価を行った．まず，データセット B1 における前半 30 分のログを学習用データ，データセット B1–B4 における後半 30 分のログを検査用データとして検知精度の比較を行った．本実験手順では各分類モデルを再学習なしで用いた場合の検知精度を評価している．つまり，データセット B1 によって学習され分類モデルを，数日後に収集された広告リクエストで構成された検査用データセットに対して再学習なしで適用している．次に，データセット B1–B4 における前半 30 分のログを学習用データ，後半 30 分のログを検査用データとして検知精度の比較を行った．本実験手順では各分類モデルを再学習ありで用いた場合の検知精度を評価している．つまり，分類モデルがデータセットの前半 30 分のログにより毎週再学習された場合の検知精度を評価している．

図 5.7 に，各日付の検査用データセットを用いた際の precision, recall および AUC の値を示す．図における，実線，破線はそれぞれ再学習ありと再学習なしの場合の検知精度結果を表している．提案手法は，既存手法と比較して，precision, recall, AUC の値が安定して高い事が分かる．加えて，提案手法では再学習なし，再学習ありの両ケースにおいて，精度に大きな差が無い事が分かる．一方で既存手法の検知精度は日付によって大きく異なっており，再学習を行わない場合には時間経過によって検知精度が低下していく傾向がある事が分かる．これらの結果は，提案手法は既存手法と比較して時間経過による検知精度低下に対してロバストである事を示しており，提案手法において新たに設計された，パブリッシャとクライアントの統計に基づいた特徴量

表 5.4　検知された広告リクエスト

ログの種別	全体の件数	検知数 (割合)
インプレッションログ	8,652,857	588,274　(6.8%)
クリックログ	22,213	1,224　(5.5%)

表 5.5　ユニークなクライアント IP アドレスおよびパブリッシャー URL の数と統計値

説明	ユニーク数	平均	中央値	標準偏差
IP アドレス (全て)	2,333,456	3.71	2	131.01
IP アドレス (検知されたもの)	15,324	61.22	4	1614.36
パブリッシャー URL (全て)	2,449,307	3.42	1	76.11
パブリッシャー URL (検知されたもの)	191,358	18.44	2	265.60

は，リクエストのバースト性に基づいた特徴量と比較して安定していると言える．

5.4.4　提案手法における特徴量の貢献

提案手法と既存手法は，正規ユーザによる広告リクエストの中から攻撃者による不正なリクエストを見分ける上で利用している特性が異なっている． 5.4.2 節の評価結果から，提案手法は既存のバースト性に基づいた手法より広告不正の見逃しが少ない事を確認した．よって，提案手法で新規に設計したクライアントとパブリッシャの統計に基づく特徴量が，バースト性の大きい広告リクエストを検知する事に寄与していると考えられる．また同様に 5.4.3 の結果から，それら統計に基づいた特徴量はバースト性に基づいた特徴量より安定している事を確認した．

5.5　オンライン広告不正の大規模観測

提案手法を大規模なデータセットに適用する事で，昨今の広告不正の実態調査を行った．具体的には，5.4.2 節の評価で作成された分類モデルを，データセット A の後半のデータに対して適用し結果を分析した．なお，本評価では正解ラベルを含まない広告リクエストログに対して提案手法を適用している点に注意されたい．以下，実態調査の結果について説明する．

5.5.1　基本的な統計

表 5.4 に，検知された広告リクエスト数を示す．中央の列は調査対象であるデータセット A1 の後半のログに含まれていたインプレッションログ，およびクリックログの総数であり，最終列は，それらのうち悪性と検知された数，およびその割合である．提案手法により，8,652,857 件の広告閲覧のうち，588,274 件（6.8%）が検知された．表 5.5 は，ユニークなクライアント IP アドレスおよびパブリッシャー URL から送信された広告リクエスト数の平均値，中央地，および標準偏差の値を示している．なお，表における検知された IP アドレス，パブリッシャー URL は，少なくとも 1 件以上の悪性な広告リクエストが当該 IP アドレスもしくはパブリッシャー URL から送信されていた場合をカウントしている．検知された IP アドレスおよびパブリッシャー URL からは，検知されたなかったものより多くの広告リクエストが送信されている事を確認した．

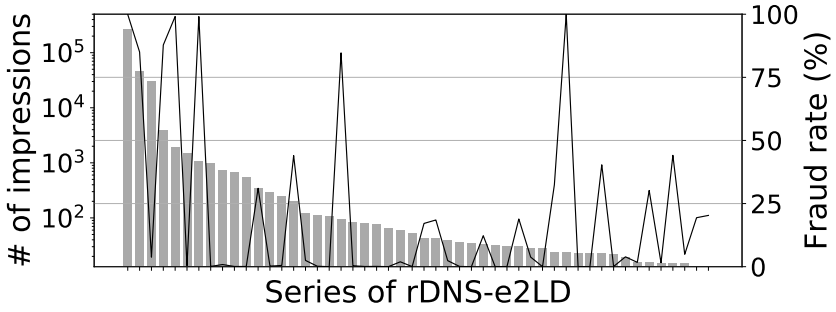

図 5.8　各 rDNS-e2LD からの不正な広告閲覧（上位 50 位）

5.5.2　クライアントの IP アドレスに関する分析

各 rDNS-e2LD から送信された広告リクエストの内訳について分析した．本調査では，2,641 件の rDNS-e2LD を観測した．図 5.8 に，検知された広告リクエストの送信数が多い rDNS-e2LD の上位 50 位までを示す．図における棒グラフは，共通する rDNS-e2LD を持つクライアント IP アドレスから送信された不正な広告リクエストの総数を表しており，折れ線グラフは共通する rDNS-e2LD をもつクライアント IP アドレスから送信された全ての広告リクエストのうち検知されたものの割合（悪性率）を表している．なお，棒グラフにおける y 軸はログスケールである点に注意されたい．分析の結果，特定の rDNS-e2LD から大量の不正な広告リクエストが送信されている事を確認した．特に，調査全体で検知された不正な広告リクエストのうち，45% が単一の rDNS-e2LD から送信されていた．加えて，いくつかの rDNS-e2LD にでは悪性率が突出して高い事を確認した．特に不正な広告リクエストの検知数が多かった上位 50 位までの rDNS-e2LD のうち，7 つの rDNS-e2LD において，悪性率が 80% を超えていた．この結果から，共通的な rDNS-e2LD を持ついくつかの IP アドレスのグループは，広告不正によって悪用されやすい特性があると言える．表 5.6 は，悪性な広告リクエストの多かった上位 20 位までの rDNS-e2LD における統計を表している．なお，実際の rDNS-e2LD の値は省略している．上位 20 位までの rDNS-e2LD を手動で分析し，それらの rDNS-e2LD に共通する IP アドレス郡を管理する組織，および利用目的を調査した．表 5.6 における最終列が “クラウドサービス” となっている場合，当該 rDNS-e2LD に共通する IP アドレス郡がクラウドや VPS などの仮想マシンを提供するサービスによって管理されている事を表している．同様に，“プロキシサービス”，“モバイルキャリア”，“ISP” となっている場合，当該 rDNS-e2LD に共通する IP アドレス郡がそれぞれ Web プロキシを提供するサービス，モバイルネットワークオペレータ，および ISP によって管理されている事を表している．クラウドサービスによって管理されている rDNS-e2LD において，悪性率が突出して高い傾向がある事から，これらのサービス攻撃者によって悪用されやすい事が示唆される．また，プロキシサービスによって管理されている rDNS-e2LD_2 において，同様に悪性率が高い事から，攻撃者が広告不正を行う際に当該プロキシサービスを用いて，送信元 IP アドレスを分散させている事が示唆される．

5.5.3　パブリッシャの FQDN に関する分析

パブリッシャー URL の FQDN ごとの広告リクエストの内訳について分析した．本調査では，113,860 件の FQDN が観測された．図 5.9 に，検知された広告リクエストの送信数が多い FQDN の上位 50 位までを示

表 5.6　各 rDNS-e2LD から送信された悪性な広告リクエストの統計（上位 20 位）

rDNS-e2LD	検知されたインプレッションログの件数 (悪性率)		ユニークなクライアント IP アドレスの数 (悪性率)		ユニークなクライアントの数	rDNS-e2LD のカテゴリ
rDNS-e2LD_1	269,214	(99.9%)	339	(77.9%)	418	クラウドサービス
rDNS-e2LD_2	45,539	(85.0%)	129	(93.0%)	16,779	プロキシサービス
rDNS-e2LD_3	30,284	(3.8%)	162,843	(6.3%)	345,176	モバイルキャリア
rDNS-e2LD_4	3,932	(87.8%)	65	(21.5%)	150	クラウドサービス
rDNS-e2LD_5	1,891	(99.1%)	3	(100.0%)	184	クラウドサービス
rDNS-e2LD_6	1,486	(0.4%)	119,286	(1.0%)	125,898	ISP
rDNS-e2LD_7	1,057	(99.1%)	1	(100.0%)	97	クラウドサービス
rDNS-e2LD_8	1,002	(0.1%)	321,989	(0.2%)	343,412	ISP
rDNS-e2LD_9	739	(0.9%)	5,939	(5.6%)	27,835	ISP
rDNS-e2LD_10	677	(0.1%)	273,584	(0.2%)	288,615	ISP
rDNS-e2LD_11	539	(0.1%)	210,626	(0.1%)	231,427	ISP
rDNS-e2LD_12	351	(31.1%)	6	(100.0%)	90	ISP
rDNS-e2LD_13	292	(0.2%)	42,945	(0.4%)	45,592	ISP
rDNS-e2LD_14	244	(0.5%)	10,551	(1.2%)	16,998	ISP
rDNS-e2LD_15	202	(44.0%)	180	(53.9%)	205	クラウドサービス
rDNS-e2LD_16	121	(2.5%)	609	(2.6%)	1,304	ISP
rDNS-e2LD_17	112	(0.2%)	20,300	(0.2%)	22,616	ISP
rDNS-e2LD_18	107	(<0.1%)	75,802	(0.1%)	338,795	モバイルキャリア
rDNS-e2LD_19	94	(84.7%)	67	(95.5%)	92	ISP
rDNS-e2LD_20	85	(0.4%)	2,297	(2.0%)	7,080	ISP

す．なお，図における棒グラフと折れ線グラフは図 5.8 と同様の値を表している．悪性率が極端に高い 2 つの FQDN が確認された．また，上位 50 位のうち 41 の FQDN で悪性率が 10 %を超えており，各 FQDN において一定の割合で広告不正が行われている傾向が確認された．表 5.7 は，悪性な広告リクエストの多かった上位 20 位までの FQDN における統計を表している．なお，実際の FQDN の値は省略している．表 5.7 における最終列は，上位 20 位までの FQDN を手動で分析し，それらの FQDN の管理組織，および利用目的について調査した結果を表している．FQDN_11 と FQDN_20 の悪性率が，その他の FQDN と比較して極めて高い事を確認した．また，手動での調査の結果，FQDN_11 は Web コンテンツの配信などに利用されるホスティングサービスによって管理されていた．加えて，FQDN_11，および FQDN_20 を含むパブリッシャ URL を手動で分析した結果，コンテンツにアドタグのみを含む大量の URL が存在する事を確認した．これは，当該 URL へブラウザでアクセスを行うと，画面上に広告のみが表示される事を意味している．FQDN_3, FQDN_7 および FQDN_15 はリワードサイトによって利用されていた．リワードサイトは，ユーザは特定のタスク（アンケートへの回答，特定のサービスでのアカウント登録，ゲーム など）を実施することで換金可能なポイントを得る事ができるサービスである．このようなタイプの Web サイトでは，タスクを実施するページ上に大量の広告が表示されているケースが多い．このようなリワードサイトからの広告リクエストが検知されている事か

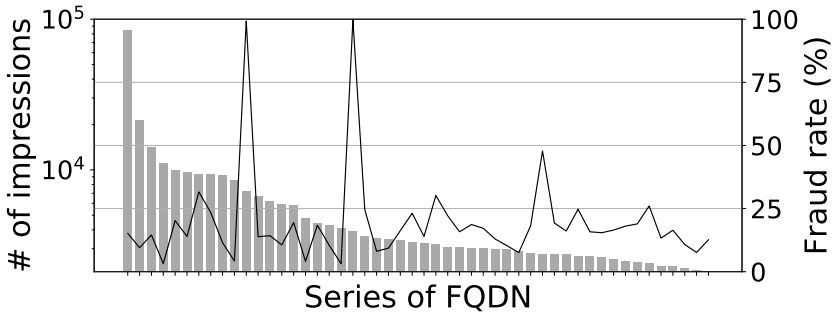

図 5.9　各 FQDN における不正な広告閲覧（上位 50 位）

ら，エンドユーザがこれらのサイトで与えられたタスクを自動化し，機械的なアクセスを行っている事が示唆される．リワードサイトは，サービスがユーザの同意を得ており，かつタスクを実施するページで不適切な広告表示を行っていない限り，それ自体が不正な Web サービスであるとは言えないが，一方でそれらのサービスデザインは広告不正を誘発していると考えられる．本調査では，上記のカテゴリの他にも複数の Web サイト（例：ニュースサイト，ブログサービス など）が広告不正に利用されている事例が確認された．

5.5.4　クライアントのアクセス環境に関する分析

広告リクエストの送信元クライアントにおける OS のバージョンの内訳について分析した．本調査では，56 種類の OS バージョンが確認された．表 5.8 は，悪性な広告リクエストの多かった上位 20 位までの OS バージョンにおける統計を表している．今回調査したデータにおいては，Windows XP から大量の不正な広告リクエストが送信されている事を確認した．具体的には，本調査で検知された全ての不正な広告リクエストのうち，83% が Windows XP から送信されていた．このような Windows XP から送信されている広告リクエストは，ユーザエージェントの値を固定して広告リクエストを送信する不正なプログラム（例：ボット）に起因している事が予想される．なお，提案手法はこれらの OS バージョンの特徴を検知に用いていないが，クライアン環境に関する注目すべき特徴を検知できている事は特筆すべき点である．上記に加えて，Android デバイスから多くの不正な広告リクエストが送信されているが確認された．前述の Windows XP による大量の不正な広告リクエストを除くと，今回の調査で検知された不正な広告リクエストのうち約 8 割が Android デバイスから送信されていた．これらの不正な広告リクエストは，Android を標的としたマルウェアやアドウェアに起因している場合や，攻撃者が Android 端末からのリクエストを模して不正な広告リクエストを送信している場合などが考えられる．

5.6　議論

本節では，本研究における制約事項と今後の課題，研究倫理について述べる．

5.6.1　制約事項

正解ラベルの生成方法: 本研究では，教師あり機械学習における正解ラベルとして，広告不正対策サービスのオペレータによる手動のラベル付けの結果を用いた．手動でのラベル付けの具体的なルールは非公開である

表 5.7　各 FQDN から送信された悪性な広告リクエストの統計（上位 20 位）

FQDN	検知されたインプレッションログの件数 (悪性率)		ユニークなクライアント IP アドレスの数 (悪性率)		ユニークなクライアントの数	FQDN のカテゴリ
FQDN_1	84,111	(15.2%)	157,196	(0.2%)	160,702	動画視聴サイト
FQDN_2	21,383	(9.5%)	97,459	(0.5%)	110,166	ブログサービス
FQDN_3	14,168	(14.5%)	33,422	(0.2%)	36,071	リワードサイト
FQDN_4	11,087	(3.2%)	129,560	(2.9%)	162,177	ニュースサイト
FQDN_5	9,991	(20.2%)	17,239	(0.9%)	17,346	EC サイト
FQDN_6	9,646	(14.0%)	20,584	(0.1%)	20,717	動画視聴サイト
FQDN_7	9,392	(31.6%)	4,227	(0.5%)	4,294	リワードサイト
FQDN_8	9,391	(23.4%)	16,058	(0.4%)	16,169	ニュースサイト
FQDN_9	9,173	(11.5%)	35,169	(0.3%)	36,796	ニュースサイト
FQDN_10	8,466	(4.2%)	73,825	(2.2%)	87,187	フォーラムサイト
FQDN_11	7,230	(99.3%)	258	(99.6%)	729	ホスティングサービス
FQDN_12	6,633	(13.8%)	19,190	(0.2%)	19,342	ニュースサイト
FQDN_13	6,139	(14.3%)	17,580	(0.1%)	17,682	動画視聴サイト
FQDN_14	5,946	(10.6%)	17,170	(0.4%)	18,002	フォーラムサイト
FQDN_15	5,827	(19.4%)	5,022	(0.4%)	5,107	リワードサイト
FQDN_16	4,738	(4.1%)	39,041	(0.7%)	39,842	フォーラムサイト
FQDN_17	4,411	(18.4%)	5,822	(0.5%)	5,877	動画視聴サイト
FQDN_18	4,301	(10.5%)	21,965	(1.1%)	23,947	フォーラムサイト
FQDN_19	4,091	(3.2%)	38,610	(1.6%)	47,991	フォーラムサイト
FQDN_20	3,918	(99.8%)	3	(100.0%)	12	不明

が，例えば手動でのラベル付けにおいて考慮されていると思われる特徴（例:攻撃者に特有の IP アドレス群，Web サイト URL など）が，提案手法において設計した特徴量の有効性に影響を与えていることが考えられる．そのため，例えば異なる方法で生成した正解ラベルを用いた場合に，本研究における評価と異なる結果となる可能性がある．このような正解ラベルの生成方法に起因する問題を踏まえると，例えば教師無し機械学習を用いた検知手法の検討など，正解ラベルに依存しない手法の検討が今後の課題である．

データセットの偏り: 本研究で分析されたデータセットには，観測点や観測期間に起因するバイアスが存在する可能性がある．表 5.9 は，本調査で観測した広告リクエストの送信元の国情報について上位 5 カ国を示している．本調査で観測した広告リクエストのうち，95.3% は日本からの広告リクエストであった．これは，本研究で行った調査の結果は，日本における広告不正の実態を示しており，他国で同様の調査を行った場合結果が異なる可能性がある．同様に，オンライン広告のプラットフォームを提供する多数の広告ネットワークが存在しているが，今回の調査に用いたデータセットは単一の広告ネットワーク上で観測された広告リクエストのログで構成されている．そのため，本調査の結果は調査対象とした広告ネットワークにおける傾向が反映されており，別の広告ネットワークにおいて同様の調査を行った場合結果が異なる可能性がある．調査に用いたデータセットは，特定の期間の広告リクエストで構成されているため，より長期間のデータから抽出可能な広告

表 5.8　各 OS バージョンから送信された不正な広告リクエストの統計（上位 20 位）

OS バージョン	検知されたインプレッションログの件数 (悪性率)		ユニークなクライアント IP アドレスの数 (悪性率)		ユニークなクライアントの数
WindowsXP	493,233	(95.1%)	8,171	(0.9%)	8,212
Android7	30,948	(2.4%)	423,298	(0.7%)	543,686
Android5	16,770	(2.7%)	226,423	(1.8%)	269,125
Android4	14,650	(2.4%)	225,483	(1.7%)	280,049
Android6	12,169	(1.4%)	290,507	(0.3%)	362,672
Windows7	4,575	(0.4%)	345,008	(0.4%)	356,542
Windows10	4,242	(0.2%)	579,575	(0.3%)	602,586
iOS9	2,884	(4.1%)	26,943	(3.0%)	28,580
iOS10	1,836	(0.3%)	200,804	(0.5%)	217,204
iOS8	1,083	(4.5%)	9,249	(1.0%)	9,715
Windows8.1	941	(0.3%)	109,177	(0.2%)	111,147
MacOSX10	686	(0.7%)	32,804	(0.2%)	33,440
SymbianOS9	568	(93.1%)	31	(96.8%)	107
iOS11	533	(0.1%)	176,873	(0.2%)	185,747
Android8	526	(5.3%)	4,209	(0.7%)	4,404
WindowsPhone10	414	(51.6%)	178	(11.2%)	186
iOS5	379	(22.1%)	573	(5.2%)	584
Windows8	318	(2.4%)	4,854	(1.0%)	4,910
BlackBerryOS7	223	(94.1%)	24	(83.3%)	60
ChromeOS9765	217	(50.2%)	169	(62.7%)	169

表 5.9　国情報の統計 (データセット A)

国名	広告リクエスト数（割合）	
日本	13,165,257	(95.3%)
アメリカ	459,331	(3.3%)
台湾	91,049	(0.7%)
韓国	3,090	(<0.1%)
不明	76,210	(0.6%)

不正の特徴を網羅していない可能性がある．より長期間に観測された広告リクエストの分析は今後の課題である．

クライアント識別の粒度: 今回，個々のクライアントの識別を行う際には，IP アドレスとユーザエージェントの組み合わせを用いた．本アプローチは，Network Address Translation (NAT) や HTTP proxy の存在により，IP アドレス配下に同一ユーザエージェントのクライアントが複数存在する場合に誤検知を生じる．これらを解消するには，Cookie やその他のブラウザフィンガープリンティング技術により，より細かい粒度でクライ

アントを識別する必要がある．また，トラッキング自体が攻撃者によって回避されてしまう可能性もある．これらを考慮した上で，より誤検知を生じにくい適切な粒度でクライアントを識別する事は今後の課題である．

5.6.2 研究倫理

本研究は，エンドユーザ，パブリッシャー，広告主などオンライン広告に関わるステークホルダーのプライバシーや研究倫理に配慮して行われた．調査に用いたデータセットには，広告主を特定する情報（例：広告主の名前や広告コンテンツ など）が含まれていない． 5.3.2 節で説明したとおり，本調査ではデータセットに含まれるクライアントの IP アドレスを直接利用せず，クライアント IP アドレスに関する情報を抽出した後に匿名化した上で分析を実施した．また，特定のユーザの属性を追跡するような分析は行われなかった．加えて，IP アドレスの逆引き結果やドメイン名の具体的な値は本論文では省略された．本研究での多くの分析はオフラインで行われたが，いくつかの分析の中で特定の Web サイトへのアクセスを行った．その際，各 Web サイトにおける Acceptable use policy (AUP) に抵触しないよう配慮してアクセスを行った．また，各種サーバへの過負荷を避ける為に限られた回数のアクセスで分析を実施した．

5.7 まとめと今後の課題

本章では，広告ネットワーク上で観測される不正な広告リクエストを検知する手法を提案した．提案手法において，攻撃者が正規ユーザからの広告リクエストにおけるクライアント環境やパブリッシャー Web サイトの統計を知り得ないという仮定に基づいた新たな特徴量を設計した．提案手法の評価の結果から，新たに設計された特徴量を用いる事で従来手法における広告リクエストのバースト性に基づいた特徴量を用いる場合より，検知精度の指標である recall の値が 10% 改善される事を確認した．また，提案手法を実際の広告ネットワーク上で観測された大規模な広告リクエストログに対して適用する事で広告不正の実態調査を行った．調査の結果から，約 800 万件の広告リクエストのうち 6.8% が広告不正に起因している事が確認されると共に，昨今の広告不正における特筆すべき複数の特徴を明らかにした．具体的には，大量の不正な広告リクエストが，クラウドサービスによって管理されている IP アドレス郡から送信んされている事例や，WindowsXP や Android など特定の OS バージョンのクライアントから大量の不正な広告リクエストが送信されている事例などを観測した．上記のような事例は最新の広告不正の特性を反映しており，得られた知見は今後より高度な広告不正検知技術を設計する上で活用可能である．

第 6 章

アプリ改ざんに起因する不正なマネタイズへの対策フレームワーク

6.1 はじめに

攻撃者が Android アプリの改ざんにより大量のマルウェア作成しそれらを用いた不正なマネタイズにより利益を得ている背景には，アプリの自動的な改ざんという低コストな攻撃により大きな利益を得ることができるという課題が存在している．本研究では，攻撃者の攻撃にかかるコストを増加させ，かつ攻撃によって得られる利益を減少させる事で，攻撃者による攻撃を非効率なものとして金銭的利益を目的とした攻撃を抑制する対策フレームワークについて検討を行った．検討の結果，自動的なアプリの改ざんを防止する対策技術とオンライン広告における網羅的かつ対策回避が困難な広告不正検知技術の組み合わせにより，アプリ改ざんに起因する不正なマネタイズへの対策フレームワークを構築可能であり，アプリの流通やマネタイズを支えているエコシステムの健全化に貢献可能であると考える．

本章では，本研究の提案手法を組み合わせる事で実現するアプリ改ざんに起因する不正なマネタイズの脅威への対策フレームワークの詳細について述べる．また，対策フレームワークを有効とする攻撃の特性について論じると共に，提案フレームワークを実世界に社会実装する上での課題について述べる．

6.2 アプリ改ざんに起因する不正なマネタイズへの対策フレームワーク

アプリ改ざんに起因する不正なマネタイズへの対策フレームワークの概要について述べる．図 6.1 に提案する対策フレームワークの全体像を示す．提案する対策フレームワークは，攻撃者がアプリ改ざんおよびマルウェアの拡散にかかるコストを増加させる**アプリ改ざん対策部分**と，攻撃者が攻撃によって得られる利益を減少させる**不正マネタイズ対策部分**によって構成される．アプリ改ざん対策部分は，4 章にて提案した，アプリの自動的な改ざん防止技術を，アプリの開発および流通を支えているインフラストラクチャーであるアプリマーケット中に組み込む形で実現する．不正なマネタイズの防止部分は，5 章にて提案したオンライン広告不正検知技術を，アプリのマネタイズを支えているインフラストラクチャーである広告ネットワーク上に組み込む形で実現する．

提案フレームワークの導入後に攻撃者がアプリの改ざんにより不正なマネタイズを行おうとした際のフレームワークの効果について説明する．開発者によって作成された正規アプリは，アプリマーケット上にて自動的な改ざんへの対策がなされた上で流通する．この際，アプリ改ざん対策は自動的に付与されるため，個々の開

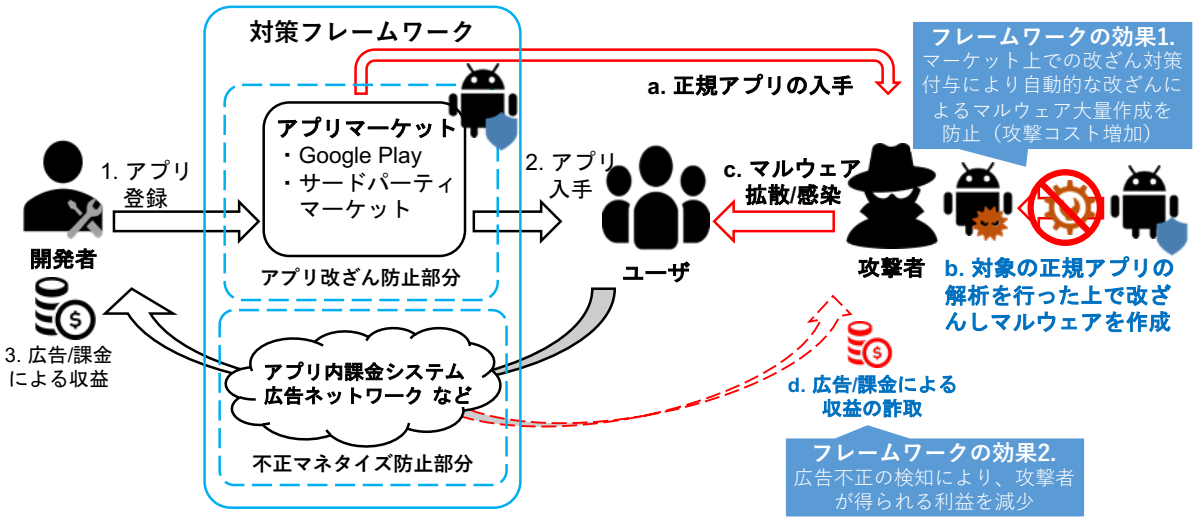

図 6.1　アプリ改ざんに起因する不正なマネタイズへの対策フレームワークと効果

発者の知識やスキルに依存せず堅牢な改ざん対策が付与可能である．また，各アプリに付与される改ざん対策は，4 章で提案した方式によりランダム性を有しているため，それぞれのアプリに対して共通的な処理によって改ざん対策を無効化する事は困難である．そのため攻撃者がマーケット上から人気のある正規アプリ入手し，改ざんによりマルウェアを作成しようとした場合，自動的な処理で改ざんを行うことができずそれぞれのアプリに特化した解析を行い改ざん対策を無効化した上で目的の改ざんを行う必要がある．これにより，攻撃者が大量の人気アプリを改ざんしマルウェアを作成する際には，改ざん対象とするアプリの数に比例したコストを強いられる事になる．同時に，仮に攻撃者がコストをかけて人気アプリを改ざんし，不正なマネタイズを行おうとした場合でも提案フレームワークにおける不正なマネタイズ防止部分によって得られる利益を減少させる事が可能である． 5 章で提案した方式では，広告不正によって発生するリクエストのバースト性に依存せず検知が可能である．そのため，攻撃者がアプリ改ざんによってマルウェアを多数作成し，分散的に広告不正を行った場合でもそれを検知し不正なマネタイズを防止する事ができる．また， 5 章で提案した方式では，広告不正の検知精度が時間経過によって低下しにくいため，防御側は少ないコストで継続的に攻撃を検知可能であるメリットもある．

上記の通り，提案フレームワークにおける改ざん対策部分によって，攻撃者がマルウェアを作成し拡散させる際に発生するコストが上昇する．同時に，仮に攻撃者がコストをかけて大量のマルウェアを作成し拡散させた場合にも，対策フレームワークにおける不正なマネタイズ対策部分において得られる収益を減少させる事が可能である．このように，対策フレームワークの導入により，攻撃者は攻撃コストの増加と得られる利益の減少を同時に強いられる事になるため，アプリ改ざんを用いた攻撃は非効率な攻撃手法となり，攻撃を止めざるを得ない状況に追い込む事が可能である．

6.3 アプリ改ざんに起因する不正なマネタイズへの対策フレームワークの構成要素

本節では，提案するアプリ改ざんに起因する不正なマネタイズへの対策フレームワークの構成要素の役割について説明する．

6.3.1 アプリ改ざん対策部分（3 章・4 章）

アプリ改ざん対策部分では，マーケット上に登録されるアプリに対して網羅的に改ざん検知機能を付与する．また，付与される改ざん検知機能の実装は，アプリごとに異なっておりランダム性を有する．アプリの流通経路であるマーケット上で網羅的な対策付与が行われるため，攻撃者は対策を付与されていない人気のアプリを入手する事が困難になる．また，攻撃者が本対策によって保護されたアプリを改ざんしようとした場合には，個々のアプリにごとに異なる改ざん検知機能の実装を詳細に解析した上で全ての改ざん検知機能を無効化する必要がある．以上より，攻撃者がマーケット上から大量のアプリを入手し自動的な改ざんによって大量のマルウェアを作成する事を困難にし，攻撃にかかるコストを増加させる．

6.3.2 不正マネタイズ防止部分（5 章）

不正マネタイズ防止部分では，攻撃者によって行われる分散的な広告不正攻撃を検知し不正なマネタイズを抑制する．攻撃者が分散的な広告不正によって大きな利益を得ている背景には，大量の感染端末によって構成される攻撃用のインフラストラクチャを用意した上で個々の感染端末から発生する広告リクエストの数を小さくする事で，既存技術による検知を回避して長期的に攻撃を行っている事が想定される．そのため不正マネタイズ防止部分における広告不正検知は，発生する広告リクエストの量やバースト性に依存せず分散型の広告不正攻撃を検知する．これにより感染端末を用いた大規模な広告不正を防止し，攻撃者が得られる利益を減少させる．

6.4 提案するフレームワークの有効性の根拠となる攻撃の前提

攻撃者が Android マルウェアを用いて金銭的な利益を得る攻撃には，例えばバンキングマルウェアやクレジットカード情報の詐取を用いた不正送金のような，少数の感染端末から大きな利益を得る攻撃と，個々の感染端末からは小さな利益しか得られないが，多くのユーザに対してマルウェアを感染させる事で結果として大きな利益を得る攻撃に大別される．本研究で着目しているアプリ改ざんによる不正なマネタイズの攻撃は後者の攻撃に分類される．このような個々のマルウェア感染による利益は小さいが広く感染を拡大させる事で利益を得る攻撃が成り立つのは，それらの攻撃が以下の点を前提にしている為である．

- 低コストにマルウェア感染を拡大させる事が可能である点
  攻撃者が不正なマネタイズを行う際には，正規のアプリが利用しているマネタイズの仕組みを悪用する方法が取られる．一方で，一般的にインストール数が少なく少数のユーザにしか利用されないアプリによるマネタイズの効果は小さく開発者は大きな収益を得ることはできない．これは攻撃者がマルウェアにより不正なマネタイズを行う場合も同様である．そのため，攻撃者にとって多数のユーザに利用され

大きな収益を得ることが可能な人気のアプリを作成するよりも，低コストにマルウェア感染を拡大させる事で得られる利益の方が大きく無いと攻撃は成り立たないと言える．

- 正規アプリによるマネタイズとマルウェアによるマネタイズを区別しにくい点
  マルウェアによる不正なマネタイズの活動は，ユーザが正規アプリを利用する事で行われる通常のマネタイズの活動の中に紛れる事で行われる．なぜならば，そもそもマルウェアによる不正なマネタイズの活動と正規アプリによる通常のマネタイズの活動が高精度に区別可能である場合には攻撃は成り立たない為である．

上記のような前提を踏まえると，攻撃者が自動的な改ざんへの対策が付与された個々のアプリに対して無限にコストをかけて改ざんを行う事は考えにくい．なぜならば，攻撃の前提として低コストにマルウェア感染を拡大させる必要があり，そのような高コストな攻撃は得られる利益に見合わないためである．よって，自動的な改ざんへの対策によって攻撃者のコストを増加させる事で，コストと利益のトレードオフにより攻撃を抑制できると考えられる．同様に，攻撃者が不正なマネタイズを行う上で，攻撃者は検知を回避する為に，不正な広告リクエストにおけるリクエスト量やパラメータなどを正規の広告リクエストに見せかけて攻撃を行うと考えられる．しかし提案フレームワークにおける広告不正検知手法は，攻撃者の立場から知り得ない統計量に基づいた特徴量を利用して検知を行っているため，攻撃者は提案手法を回避できるような不正な広告リクエストを送信する事が困難であり，結果として攻撃による利益を減少させる事ができると考えられる．

上記のとおり，攻撃が有効となる前提を崩す事が可能であることから，提案する対策フレームワークはアプリ改ざんに起因する不正なマネタイズの脅威への対策として有効であると考えられる．

6.5 提案する対策フレームワークの社会実装に向けて

本節では，アプリ改ざんに起因する不正なマネタイズへの対策フレームワークを社会実装する上での残課題について述べる．提案フレームワークを実現する上で，アプリの流通やマネタイズに関わるステークホルダーとの連携が必須である．提案フレームワークの効果は，世の中に多数存在するアプリマーケットや広告ネットワークの多くに対策が実装されるほど大きくなる．

フレームワークにおけるアプリ改ざん防止部分の実装/運用に関わるステークホルダーとしては公式マーケットである Google Play とサードパーティマーケットが考えられる．Google Play は Android アプリの流通経路として大きなシェアを誇るため，世の中で流通するアプリにおいて対策の網羅性を高める観点で，フレームワークの実装先として最適である．一方で，攻撃者はサードパーティマーケットからアプリ入手する事も可能である．特に人気アプリの開発者は複数のマーケットにおいて同一のアプリをマルチリリースするケースが存在する事，およびサードパーティマーケット自体が，マルウェア拡散の観点から攻撃者に悪用されやすい傾向がある事から，公式マーケットだけでなく，サードパーティマーケットにおいても同時にフレームワークによる対策を実施する事が望ましい．結果として，提案フレームワークによる対策が実施されているアプリマーケットの数が増えるほど，世の中で流通するアプリが保護される割合が高くなり，攻撃者にとって改ざん対象となりうる未対策のアプリを入手する事が困難になるといえる．

フレームワークにおける不正マネタイズ防止部分の実装/運用に関わるステークホルダーとして，広告ネットワークが挙げられる．昨今のオンライン広告は，単一の広告ネットワークにより広告主とパブリッシャーの間が仲介されているケースは少なく，目的ごとに細分化された多数の組織やサービスが連携した上で成り立っている．そのため，提案フレームワークを導入する上で，オンライン広告に関わる多数のステークホルダーの

うち，誰がどの段階で対策を実施するべきか今後検討が必要である．特に，近年では広告ネットワークからの委託を受けて広告不正対策を専門に取り組む企業 [3] も出現しており，オンライン広告におけるサービスの細分化が進んでいる事から，このような広告不正対策を専門とするようなサービスベンダとの連携により，提案フレームワークを効率的に社会実装する事が可能であると考えられる．

アプリの流通やマネタイズに関わるステークホルダーのセキュリティ対策の現状としては，例えば GooglePlay では，アプリセキュリティ向上プログラム [28] として，マーケットに登録されるアプリにセキュリティ上の問題が無いか検査し，問題が見つかった場合には開発者に対して通知を行う仕組みが導入されている．また，アプリに対する署名付与を GooglePlay 上で行う事で開発者がアプリ署名鍵の管理と保護を Google に委託する事が可能な GooglePlay アプリ署名 [27] の機能も提供されている．同様に，広告不正への対策に向けた取り組みとして，主要な広告ネットワークである Google アドセンス [29] でも，不正な広告トラフィックを排除し，広告主や広告のパブリッシャの利益を保護するための取り組みを行っている．このように本研究で提案するようなアプリマーケット上でアプリの堅牢化を行う対策や，広告ネットワーク上で広告不正を検知する対策は，現状のステークホルダーにおけるセキュリティ対策と親和性があると考えられる．

一方で，特定のサードパーティマーケット上でアプリ改ざんによって作成されたマルウェアが多数存在する事を確認した研究事例や，一部の広告ネットワークは広告の審査が不十分であるため攻撃に悪用されやすいという点も指摘されており，アプリの流通やマネタイズに関わる一部のステークホルダーにおいてはセキュリティ対策が不十分であるのが実情である．そのため，提案フレームワークを効果的に社会実装する為には，アプリ流通やマネタイズに関わるステークホルダー全体に対して対策の導入を働きかけていく事が望ましいと言える．

Android OS の開発を主導している Google は，Android アプリを誰でも無償で開発可能にしている点や，任意のアプリを端末上にインストール可能にしている点など，Android におけるプラットフォームのオープン性を重視していると考えられる．対策フレームワークの社会実装に向けては，このようなモバイルプラットフォーム全体としての潮流や技術の方向性なども踏まえて，各ステークホルダーの立場からの対策の受け入れられやすさを考慮する事も重要である．

6.6 対策フレームワークの拡張可能性

攻撃者による低コストなマルウェア拡散を防止する観点では，本研究でフォーカスしたアプリを堅牢化する対策以外にも，アプリの流通経路でのマルウェア拡散防止も有効である．GooglePlay では，Google Play Protect [26] と呼ばれるマーケット上で流通するアプリを検査する仕組みが導入されている．Google Play Protect によって，Google Play 上のアプリはマーケットへの登録時とその後のアプリ流通時に定常的に検査が行われるため，マーケット上でのマルウェア拡散が抑制される．同様に，中国のサードパーティマーケットで，アプリ対して専用のモジュールを自動的に追加した上で，アプリの流通状況をモニタリングする事が可能なサービスが提供されている例も存在する．このような流通状況のモニタリングにより，改ざんされたアプリの不正利用の状況をある程度把握する事が可能であり，被害に応じた対策を検討する上で効果的であると考えられる．提案フレームワークでは，アプリマーケット上でアプリ自身の堅牢化により流通後のアプリが改ざんされるリスクを低減するが，上記のような改ざんされたアプリの流通防止や流通後のアプリのモニタリングによる対策により提案フレームワークを拡張する事でより多層的な対策が可能である．

6.7 まとめと今後の課題

本章では，アプリ改ざんに起因する不正なマネタイズへの対策フレームワークについて提案を行った．攻撃者が改ざんによって作成したマルウェアを用いたマネタイズによって利益を得る攻撃は，個々の感染端末から得られる利益が小さいという特性があるため，大きな利益を得る為には多くの端末にマルウェアを感染させる必要がある．提案フレームワークでは，アプリマーケット上で流通するアプリに対して自動的に改ざん検知機能を付与する事で，アプリの自動的な改ざんによる大量のマルウェア作成を防止し，結果として攻撃者がマルウェア感染を拡大させる上でのコストを増加させる．同時に提案フレームワークでは，広告ネットワーク上で高精度に広告不正を検知する事で，攻撃によって得られる利益を減少させる．対策フレームワークの導入により，攻撃者は攻撃コストの増加と得られる利益の減少を同時に強いられる事になるため，アプリ改ざんを用いた攻撃は非効率な攻撃手法となり，攻撃を止めざるを得ない状況に追い込む事が可能である．提案フレームワークは，本研究で提案したアプリへの改ざん検知機能の自動的な付与技術と，広告不正検知技術を組み合わせる事で実現可能である．

提案フレームワークを実社会に実装する上で，アプリの流通やマネタイズに関わる各ステークホルダーとの密な連携が必要である．提案フレームワークの効果は，世の中に多数存在するアプリマーケットや広告ネットワークの多くに対策が実装されるほど大きくなるが，現状のアプリマーケット，および広告ネットワークの中にはセキュリティ対策が不十分なものも存在している．そのため，提案フレームワークを効果的に社会実装する為には，アプリ流通やマネタイズに関わるステークホルダー全体に対して対策の導入を働きかけていく事が望ましい．

今後，提案フレームワークの有効性を定量的に評価する為に継続的な検討を行う．また，金銭的利益を目的とした攻撃は，攻撃にかかるコストに見合った利益が得られないと成り立たないという前提，Android マルウェアによる脅威以外にも拡張する事で，その他のサイバー攻撃における対策技術についても検討を進めたい．

第 7 章

結論

本研究では，主要なモバイル OS である Android に着目し，Android マルウェアによる不正なマネタイズの防止を目的として，Android アプリの改ざんおよびオンライン広告不正に関する調査および対策手法の提案を行った．また，自動的なアプリ改ざんの防止による攻撃コストの増加，及び対策回避が困難な広告不正検知による攻撃によって得られる利益の減少を攻撃者に強いる事で，金銭的利益を目的とした攻撃を抑制可能な対策フレームワークを提案した．

本研究では，大量の Android マルウェアがアプリの自動的な改ざんによって作成されている事を受けて，既存の正規アプリが攻撃者による自動的なリパッケージに対してどの程度耐性を有しているか評価を実施した．評価では，実際に攻撃者によって用いられたリパッケージ手法を模して正規アプリに対して自動的な改ざんを行うと共に，改ざん後のアプリにおいて挿入されたコードと元のアプリのコードの両方が正常に動作するかを確認した．結果，多くのアプリで元の機能を保持したまま全自動でリパッケージが可能である事が確認した．以上より，現状の Android アプリの多くは自動リパッケージへの耐性が不十分であり，耐タンパー技術等を用いたリパッケージ対策が必要であることを示した．

次に，アプリマーケット上で Android アプリに改ざん検知機能を自動的に付与するシステムの構築方式について提案した．提案システムによりマーケット上で流通するアプリが自動的な改ざんに対して堅牢となり，アプリ開発者の知識やスキルに依存せず一括で対策付与が可能であると考えられる．また，提案システムでは，アプリに対してランダム性を有した自己改ざん検知コードを多数挿入する事で，アプリ改ざんへの堅牢性を増加させた．このような対策により，攻撃者によるアプリの自動的な改ざんが困難となり，改ざんによるマルウェアの大量生成を防止する事が可能であると考えられる．

また，攻撃者による不正なマネタイズ手段である広告不正に着目し検知技術を提案した．提案手法は，攻撃者によってコントロールが困難な特徴量に基づいて広告不正を検知しており，評価の結果，従来までの広告リクエストのバースト性に基づいた検知手法と比較して高精度に広告不正を検知可能である事を確認した．また，提案手法を用いた広告不正の実態調査によって，攻撃者がクラウドサービスに管理されている IP アドレスから大量の攻撃を行っている事例や，Windows XP や Android など，特定の OS 環境からの攻撃が多発している事例など，最新の広告不正における攻撃の特性を明らかにした．

最後に，アプリ改ざんに起因する不正なマネタイズの脅威への対策フレームワークを提案した．提案フレームワークは，自動的なアプリ改ざんを困難にする事で攻撃にかかるコストを増加させると同時に，検知回避が困難な広告不正検知手法により攻撃によって得られる利益を減少させる．これにより攻撃者にとってアプリ改ざんによる不正なマネタイズは非効率なものとなり攻撃を辞めざる負えない状況に追い込む事が可能である．アプリ改ざんによって作成したマルウェアによる不正なマネタイズの攻撃が成り立つのは，攻撃者が改ざんに

より低コストにマルウェア感染を拡大させる事ができる点や不正なマネタイズの活動が正規のマネタイズの活動と容易に区別できない点を前提にしている．提案フレームワークの導入によりこれらの前提を崩す事が可能であり，脅威への対策として有効であると言える．今後，提案フレームワークを社会実装する上で，アプリの流通やマネタイズに関わるステークホルダーとの連携を検討していく．攻撃者は，対策が不十分なアプリマーケットや広告ネットワークを悪用することが予想されるため，提案フレームワークを効果的を高める上で，アプリ流通やマネタイズに関わるステークホルダー全体に対して対策の導入を働きかけていく事が望ましいと言える．同時に，金銭的利益を目的とした攻撃が攻撃にかかるコストに見合った利益が得られないと成り立たないという前提に基づいた対策技術を，Android マルウェアにおける脅威以外にも拡張する事を検討する．

謝辞

本研究を進めるにあたり，ご支援を賜りました全ての皆様に深く感謝致します．

特に，多大なご指導とご助言を頂きました，横浜国立大学大学院環境情報研究院 松本勉教授，吉岡克成准教授に深く感謝致します．加えて，本研究に関する活発なご意見を頂きました，横浜国立大学先端科学高等研究院 田辺瑠偉特任助教に深く感謝いたします．

本論文の審査員をお引き受け頂き，有意義なご助言を頂きました横浜国立大学大学院環境情報研究院 森辰則教授，四方順司教授，白川真一講師に深く感謝致します．

本研究を進める上で議論に協力して頂きました，松本研究室，四方研究室，吉岡研究室の皆様に深く感謝致します．また，研究活動に際し多大なご援助を頂きました成松美央秘書，石舘知子技術補佐員，川村恵美子技術補佐員，高山宏明研究員に深く感謝致します．

本研究を進める上で多大なご支援を賜りました，秋山満昭氏，千葉大紀氏をはじめとした日本電信電話株式会社の皆様に深く感謝致します．

参考文献

[1] Alexa Top Sites. https://www.alexa.com/topsites.

[2] MaxMind | GeoIP2 Databases. https://www.maxmind.com/en/geoip2-databases.

[3] Momentum Inc. http://www.m0mentum.co.jp/.

[4] Mozilla Foundation | Public Suffix List. https://publicsuffix.org/.

[5] scikit-learn. http://scikit-learn.org/.

[6] The Interactive Advertising Bureau | IAB internet advertising revenue report. https://www.iab.com/wp-content/uploads/2019/05/Full-Year-2018-IAB-Internet-Advertising-Revenue-Report.pdf.

[7] ua-parser/uap-python. https://github.com/ua-parser/uap-python.

[8] What Is An Untrustworthy Supply Chain Costing The U.S. Digital Advertising Industry? https://www.iab.com/wp-content/uploads/2015/11/IAB_EY_Report.pdf.

[9] A. Annie. The State of Mobile in 2020: The Key Stats You Need to Know. https://www.appannie.com/en/insights/market-data/state-of-mobile-2020-infographic/.

[10] Apple. App Store. https://www.apple.com/jp/ios/app-store/.

[11] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, page 259–269. ACM, 2014.

[12] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[13] Broadcom. MDK: 中国で最大のモバイルボットネット. https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=36023dfa-8b66-4875-8ba0-96824f6d7cfc&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments.

[14] X. Cai and J. Heidemann. Understanding block-level address usage in the visible internet. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 99–110. ACM, 2010.

[15] H. Cho, J. Lim, H. Kim, and J. H. Yi. Anti-debugging scheme for protecting mobile apps on android platform. *The Journal of Supercomputing*, 72(1):232–246, 2016.

[16] J. Crussell, R. Stevens, and H. Chen. Madfraud: Investigating ad fraud in android applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 123–134. ACM, 2014.

[17] N. Daswani and M. Stoppelman. The anatomy of clickbot.a. In *Proceedings of the First Conference on First*

Workshop on Hot Topics in Understanding Botnets, HotBots'07, pages 11–11. USENIX Association, 2007.

[18] V. Dave, S. Guha, and Y. Zhang. Measuring and fingerprinting click-spam in ad networks. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 175–186. ACM, 2012.

[19] V. Dave, S. Guha, and Y. Zhang. Viceroi: Catching click-spam in search ad networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 765–776. ACM, 2013.

[20] A. Desnos. GitHub - androguard/androguard. https://github.com/androguard/androguard.

[21] A. Developers. Shrink Your Code and Resources | Android Studio. https://developer.android.com/studio/build/shrink-code.html.

[22] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, page 393–407. USENIX Association, 2010.

[23] FireEye. Kemoge: Another Mobile Malicious Adware Infecting Over 20 Countries. https://www.fireeye.com/blog/threat-research/2015/10/kemoge_another_mobi.html.

[24] Google. Common Intent | Android Developers. https://developer.android.com/guide/components/intents-common.

[25] Google. Google Play. https://play.google.com.

[26] Google. Google Play Protect. https://www.android.com/intl/ja_jp/play-protect/.

[27] Google. Google Play アプリ署名を使用する - Play Console ヘルプ. https://support.google.com/googleplay/android-developer/answer/7384423.

[28] Google. アプリセキュリティ向上プログラム | Android Developers. https://developer.android.com/google/play/asi.

[29] Google. 無効なトラフィックを排除するための Google の取り組み - AdSense ヘルプ. https://support.google.com/adsense/answer/1348752.

[30] Goole. Android Debug Bridge (adb) | Android Developers. https://developer.android.com/studio/command-line/adb?hl=ja.

[31] Goole. Android NDK | Android Developers. https://developer.android.com/ndk.

[32] iBotPeaches. Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps. https://ibotpeaches.github.io/Apktool/.

[33] Y. Ishii, T. Watanabe, F. Kanei, Y. Takata, E. Shioji, M. Akiyama, T. Yagi, B. Sun, and T. Mori. Understanding the security management of global third-party android marketplaces. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics*, WAMA '17, page 12–18. ACM, 2017.

[34] JesusFreke. GitHub - JesusFreke/smali: smali/baksmali. https://github.com/JesusFreke/smali.

[35] Y. Jin, E. Sharafuddin, and Z. Zhang. Identifying dynamic ip address blocks serendipitously through background scanning traffic. In *Proceedings of the 2007 ACM CoNEXT Conference*, CoNEXT '07, pages 4:1–4:12. ACM, 2007.

[36] Kaspersky. Mobile malware evolution 2018. https://securelist.com/mobile-malware-evolution-2018/89689/.

[37] Y. Kikuchi, H. Mori, H. Nakano, K. Yoshioka, T. Matsumoto, and M. Van Eeten. Evaluating malware mitigation by android market operators. In *Proceedings of the 9th USENIX Conference on Cyber Security Experimentation and Test*, CSET'16, page 4, USA, 2016. USENIX Association.

[38] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, page 280–291. IEEE, 2015.

[39] lohan. AntiLVL - Android License Verification Library Subversion. `http://androidcracking.blogspot.jp/p/antilvl_01.html`.

[40] L. Luo, Y. Fu, D. Wu, S. Zhu, and P. Liu. Repackage-proofing android apps. In *Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '16, pages 550–561, 2016.

[41] E. Mariconti, L. Onwuzurike, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. In *Proceedings of the 24st Annual Network and Distributed System Security Symposium*, NDSS '17. The Internet Society, 2017.

[42] A. Metwally, D. Agrawal, and A. E. Abbadi. Duplicate detection in click streams. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 12–21. ACM, 2005.

[43] A. Metwally, D. Agrawal, and A. E. Abbadi. Detectives: Detecting coalition hit inflation attacks in advertising networks streams. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 241–250. ACM, 2007.

[44] T. Micro. A Look at Repackaged Apps and their Effect on the Mobile Threat Landscape - TrendLabs Security Intelligence Blog. `http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-into-repackaged-apps-and-its-role-in-the-mobile-threat-landscape/`.

[45] C. Mulliner, W. Robertson, and E. Kirda. Virtualswindle: An automated attack against in-app billing on android. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, page 459–470. Association for Computing Machinery, 2014.

[46] Oracle. jarsigner. `https://docs.oracle.com/javase/jp/8/docs/technotes/tools/windows/jarsigner.html`.

[47] P. Pearce, V. Dave, C. Grier, K. Levchenko, S. Guha, D. McCoy, V. Paxson, S. Savage, and G. Voelker. Characterizing large-scale click fraud in zeroaccess. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 141–152. ACM, 2014.

[48] M. Protsenko, S. Kreuter, and T. Müller. Dynamic self-protection and tamperproofing for android apps using native code. In *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security*, ARES '15, pages 129–138, 2015.

[49] P. Solutions. Java and Android Obfuscator and Protector. `https://www.preemptive.com/products/dasho/overview`.

[50] B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna. Understanding fraudulent activities in online ad exchanges. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 279–294. ACM, 2011.

[51] T. Tian, J. Zhu, F. Xia, X. Zhuang, and T. Zhang. Crowd fraud detection in internet advertising. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1100–1110. International World

Wide Web Conferences Steering Committee, 2015.

[52] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan. Soot-a java bytecode optimization framework. In *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research*, page 13. IBM Press, 1999.

[53] N. Viennot, E. Garcia, and J. Nieh. A measurement study of google play. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, page 221–233. ACM, 2014.

[54] M. Y. Wong and D. Lie. Intellidroid: A targeted input generator for the dynamic analysis of android malware. In *Proceedings of the 23st Annual Network and Distributed System Security Symposium*, NDSS '16. The Internet Society, 2016.

[55] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber. How dynamic are ip addresses? In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, pages 301–312. ACM, 2007.

[56] W. Yang, Y. Zhang, J. Li, H. Liu, Q. Wang, Y. Zhang, and D. Gu. Show me the money! finding flawed implementations of third-party in-app payment in android apps. In *Proceedings of the 24st Annual Network and Distributed System Security Symposium*, NDSS '17. The Internet Society, 2017.

[57] L. Zhang and Y. Guan. Detecting click fraud in pay-per-click streams of online advertising networks. In *Proceedings of The 28th International Conference on Distributed Computing Systems*, ICDCS '08, pages 77–84. IEEE, 2008.

[58] W. Zhou, Z. Wang, Y. Zhou, and X. Jiang. Divilar: Diversifying intermediate language for anti-repackaging on android platform. In *Proceedings of the 4th ACM conference on Data and application security and privacy*, pages 199–210. ACM, 2014.

[59] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, CODASPY '12, page 317–326. ACM, 2012.

[60] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, 2012.

[61] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off ofmy market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the 19st Annual Network and Distributed System Security Symposium*, NDSS '12. The Internet Society, 2012.

[62] 株式会社 DNP ハイパーテック. CrackProof for Android DEX. `https://www.hypertech.co.jp/crackproof/android/`.

[63] 吉田直樹, 吉岡克成, and 松本勉. 自己書換え型耐タンパー技術のスマートフォン環境における検証. In **電子情報通信学会技術研究報告**, volume 113. 電子情報通信学会, 2013.

[64] 総務省. 平成 28 年版 情報通信白書 | モバイル向けアプリ市場. `https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h28/html/nc122230.html`.

公表論文リスト

学会論文誌論文

1. 金井文宏，庄田祐樹，橋田啓佑，吉岡克成，松本勉，Android アプリケーションの自動リパッケージに対する耐性評価，情報処理学会論文誌 Vol.56， No.12，pp. 2275-2288，2015.
2. Fumihiro KANEI, Daiki CHIBA, Kunio HATO, Katsunari YOSHIOKA, Tsutomu MATSUMOTO and Mitsuaki AKIYAMA, “Detecting and Understanding Online Advertising Fraud in the Wild”, IEICE Transactions on Information and Systems, 2020, Volume E103.D, Issue 7, Pages 1512-1523.

本研究に関連する国際会議発表

1. Fumihiro KANEI, Daiki CHIBA, Kunio HATO and Mitsuaki AKIYAMA, "Precise and Robust Detection of Advertising Fraud", 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), 2019.

本研究に関連する研究会・シンポジウム等発表（査読なし）

1. 金井文宏，庄田祐樹，吉岡克成，松本勉，“Android アプリケーションの自動リパッケージに対する耐性評価”，電子情報通信学会情報システムセキュリティ研究会（ICSS）， 2014 年 7 月.
2. 金井文宏，千葉大紀，高田雄太，秋山満昭，八木毅，波戸邦夫，“広告ネットワーク上で観測されたユーザアクティビティの分析による広告不正の実態調査”，電子情報通信学会情報システムセキュリティ研究会（ICSS）， 2018 年 3 月.

その他

1. Hiroki Nakano, Fumihiro Kanei, Yuta Takata, Mitsuaki Akiyama and Katsunari Yoshioka. Towards Finding Code Snippets on a Question and Answer Website Causing Mobile App Vulnerabilities. IEICE Transactions on Information and Systems, 2018, Volume E101.D, Issue 11, Pages 2576–2583.
2. Tatsuhiko Yasumatsu, Takuya Watanabe, Fumihiro Kanei, Eitaro Shioji, Mitsuaki Akiyama, and Tatsuya Mori. Understanding the Responsiveness of Mobile App Developers to Software Library Updates. The 9th ACM Conference on Data and Application Security and Privacy (CODASPY), 2018.
3. 源平祐太，中川雄太，高田一樹，小出駿，金井文宏，秋山満昭，田辺瑠偉，吉岡克成，松本勉，悪性

Web サイトに到達しやすい危険検索単語の検知，情報処理学会コンピュータセキュリティシンポジウム 2019（CSS2019），2019.【優秀論文賞受賞】

4. Takuya Watanabe, Mitsuaki Akiyama, Fumihiro Kanei, Eitaro Shioji, Yuta Takata, Bo Sun, Yuta Ishii, Toshiki Shibahara, Takeshi Yagi and Tatsuya Mori. Study on the Vulnerabilities of Free and Paid Mobile Apps Associated with Software Library. IEICE Transactions on Information and Systems, 2020, Volume E103.D, Issue 7, Pages 276-291.
5. 金井文宏，長谷川彩子，塩治榮太朗，秋山満昭，セキュアなソフトウェア開発の阻害要因分析，情報処理学会コンピュータセキュリティシンポジウム 2020（CSS2020），2020.【最優秀論文賞受賞】