Computer Security Symposium 2017
23 - 25 October 2017

# モバイルアプリ開発者による脆弱性対応の実態調査

安松 達彦[1,a)]　金井 文宏[2]　渡邉 卓弥[2]　塩治 榮太朗[2]　秋山 満昭[2]　森 達哉[1]

**概要**：マーケットに存在するモバイルアプリには脆弱性を有するものが少なくないが，それらの脆弱性がいつどのようなタイミングで修正されるか，あるいは放置されたままなのかは明らかではない．本研究ではAndroid の公式マーケットで配布されているアプリを対象とし，アプリ開発者による脆弱性対応の実態調査を行う．具体的には Google の App Security Improvement Program ( ASI Program ) による脆弱性修正キャンペーンの対象となった脆弱性を持つアプリを抽出し，修正のタイミングを調査する．脆弱性への対応をキャンペーン開始以前，キャンペーン中，キャンペーン終了後に脆弱性を修正したアプリや開発者，および脆弱性を修正しなかったアプリや開発者に分類し分析を行う．過去全てのバージョンを収集できた 21,046 種類のアプリ，計 142,611 個の異なる APK ファイルを対象に分析を行った結果，ASI Program の脆弱性への対応は，キャンペーン開始以前の修正が 25.9%，キャンペーン中の修正が 38.6%，キャンペーン終了後の修正が 21.0%，未修正が 14.5%であった．未修正なアプリは最低でも計 4,622 万回インストールされており，今もユーザに利用されている可能性がある．また，あるキャンペーンの脆弱性が修正されていなかったアプリの内 63.4% がコーディングを必要としないアプリ生成サービスにより生成されたアプリだった．さらにインストール数が少ないアプリはキャンペーンに対応しにくい傾向があることも確認した．

**キーワード**：モバイルアプリ，脆弱性，ライブラリ，ソフトウェア開発

# Understanding the Vulnerability Responses by Mobile App Developers

Tatsuhiko Yasumatsu[1,a)]　Fumihiro Kanei[2]　Takuya Watanabe[2]　Eitaro Shioji[2]
Mitsuaki Akiyama[2]　Tatsuya Mori[1]

***Abstract:*** Mobile applications are distributed via market. In the Android official market there are over 3.2 million apps. It is known that a considerable number of these apps have vulnerabilities. However, it has not been studied when the vulnerabilities are fixed, or remained unfixed. To answer the question, in this work we analyzed how Android application developers responded to major vulnerabilities reported in Google's App Security Improvement Program (ASI Program). ASI program launches several campaigns that set deadlines for fixing vulnerabilities. We categorized the developer's response to the vulnerability into 4 groups: fix before the campaign, fix during the campaign, fix after the campaign and unfixed apps. We conducted the analysis against 21,046 apps that we were able to collect whole versions, which sums up to 142,611 distinct APKs. Out of the apps that were vulnerable 25.9% fixed the vulnerability before the campaign, 38.6% fixed during the campaign, 21.0% fixed after the campaign, and 14.5% were still unfixed. These unfixed apps have been installed at least 46 million times, and users may still be using it. Also for a specific campaign, 63.4% of the unfixed apps were generated using an app building service. In addition we confirmed that apps with few installs tend to react to the campaign slowly or do not react at all.

***Keywords:*** mobile apps, vulnerabilities, libraries, software development

[1] 早稲田大学 基幹理工学研究科
Waseda University
[2] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[a)] ty@nsl.cs.waseda.ac.jp

## 1. はじめに

2007 年に Apple 社の iPhone がリリースされて今年で 10 年目となるが, スマートフォンやタブレット端末を対象

としたモバイルアプリの数は依然として加速度的な増加を示している．モバイルアプリはマーケットを通じて配布されることに特徴がある．Appbrain の調査 [1] によれば 2017 年 8 月現在， Android の公式マーケット上に存在するアプリの数は 320 万を越える．1 年前の 2016 年 8 月時は約 210 万のアプリが存在しており，この 1 年だけでも実に 110 万のアプリがリリースされている．また，同調査によれば特に最近にリリースされたアプリは低品質と位置付けられるアプリの割合が増えている．

これらの膨大なアプリケーションの中には脆弱性を有するものが少なくない [2,3]．Watanabe [2] らは Android アプリを対象に脆弱性の分析を行い，無料アプリのランキング上位 5,000 のアプリにおいて 70% を越えるアプリがなんらかの脆弱性を有していること，およびそれらの脆弱性を含むアプリのうち，サードパーティライブラリ由来の脆弱性を含むアプリが 70% であることを示した．これらの先行研究ではモバイルアプリの脆弱性の起源を明らかにしているが，それらの脆弱性がどのように修正されるかは明らかではない．本研究では Android の公式マーケットで配布されているアプリを対象とし，アプリ開発者が脆弱性に対していつ，どのようなタイミングで対応しているか，およびタイミングの差異を生み出す因子に関して大規模な実態調査を行った．

本研究では Android の公式マーケットである Google Play 上で配布されているランキング上位のアプリを中心に調査対象を決定し，主要な脆弱性に対しアプリ開発者がどのタイミングで対応したかを調査する．具体的には，対象とする各アプリの過去すべてのバージョンを遡って収集し，脆弱性テストを行う．これによりアプリのどのバージョンで脆弱性が修正されたのか，あるいは修正されていないままなのかを分析することが出来る．分析する主要な脆弱性として Google が提供している App Security Improvement Program [4] が実施している脆弱性修正キャンペーンの対象となった脆弱性を対象とする．App Security Improvement Program では開発者がより安全なアプリを作成するための情報発信を行っており，特に普及している脆弱性についてはキャンペーンを実施し，マーケットからの除去を目指す．

本論文で明らかになったことは以下の通りである．

- 収集したアプリについて App Security Improvement Program の脆弱性修正キャンペーンの対象となっている脆弱性への対応を調査した．この結果キャンペーン開始前の修正が 25.9% ，キャンペーン期間中の修正が 38.6%，キャンペーン終了後の修正が 21.0 %，未修正なままのアプリが 14.5%であった．
- キャンペーン対象の脆弱性が未修正なままのアプリが現在も Google Play に存在していることを確認した．その合計のインストール回数は最低でも 4,622 万回に上り，今もユーザに利用されている可能性がある．

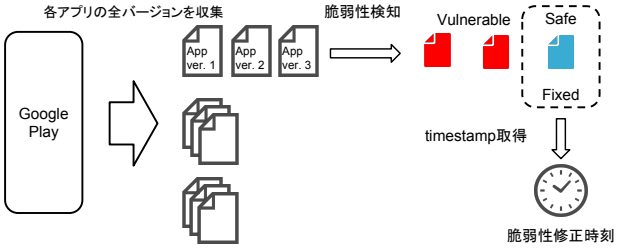


図 1　本研究での調査の概要．

- アプリ生成サービス（ Mobincube ）により生成されたアプリはキャンペーンに対応しにくい傾向にあることを示した．あるキャンペーンの脆弱性が修正されていないアプリの内 63.4% はアプリ生成サービスにより生成されたアプリであり，収集されたアプリ全体での割合 0.442% より遥かに多い．
- 生成されたアプリの内キャンペーンに対応したものについては素早く対応が行われていた．生成されたアプリのキャンペーンに対応するのにかかった平均日数は 27.4 日であり，それ以外のアプリの平均 180 日より短い．
- インストール数が少ないアプリはキャンペーンに遅れて対応する傾向にあることが分かった．キャンペーンに遅れて対応したアプリの平均インストール数は 55 万回であり，期間中に対応したアプリの 640 万回より少なくなっている．

本論文の構成は以下の通りである．2 章は調査方法の詳細を示す．3 章は調査対象としたアプリケーションおよび脆弱性を示す．4 章は調査結果のハイライトを報告する．5 章で結果から得られた知見と，本研究の制限を述べる．6 章で関連研究についてまとめ，7 章で本論文のまとめを行う．

## 2. 脆弱性対応の調査方法

本研究では調査のために過去のバージョンまで遡ったアプリの収集と脆弱性テストを行い，脆弱性への対応を調査した．本章ではアプリの脆弱性への対応の調査方法を述べる．

### 2.1 全体の概要

調査の概要を図 1 に示す．本研究での調査は大きく 3 つのステップに分かれている．まず最初に収集対象としたアプリの過去全てのバージョンを収集する．今回は Google Play のランキング上位のアプリを中心に収集対象を決めた．

次に収集したアプリに対して脆弱性検知を行った．本調査ではアプリの最新版を含めた，過去全てのバージョンを収集する．従ってどのバージョンで脆弱性が修正されているのか，もしくは修正がされていないのかを確認できる．

最後に，調査した脆弱性対応を元に分析を行う．脆弱性の修正にかかった時間，App Security Improvement Program

への対応などを分析した．

### 2.2 アプリ収集

Google Play では開発者が自由にアプリのカテゴリを選択できるようになっている．本調査では 34 種のカテゴリのランキング上位 100 位のアプリを日本向けおよび米国向けの Google Play で調査し，それらを合算した．またそれらのアプリの開発者が開発した別のアプリも調査対象とした．カテゴリ内のアプリが少なく，ランキングが 100 位まで表示されない場合は表示されたすべてのアプリを対象とした．

通常の Android 端末からはアプリの最新版のみダウンロード，インストールが可能である．しかし Google Play API を用いてリクエストを送信する際に過去のバージョン番号を指定することができる．これを利用することによってアプリの過去のバージョンをダウンロードすることができる．API で指定するバージョン番号は普段ユーザが目にする versionName (e.g., 2.2.1) ではなく，開発者が任意に指定することができる $2,100,000,000$ 以下の正の整数値， versionCode である．versionCode には多くの開発者間で共有されているような統一的な指定方法が存在せず，新しいアプリのバージョンほど大きな値でなければならないというルール以外存在しない．よって， versionCode は $2.1 \times 10^9$ 以下の正の値全体に分布していると考えられる．

アプリの過去のバージョンの versionCode を容易に知る手段は提供されていない．しかし API を利用してアプリの最新版の versionCode は知ることができる．よって本研究ではまず各アプリについて最新版の versionCode の値を調査した．この調査した最新版の値を一つ一つデクリメントしてアプリをダウンロードするリクエストを繰り返し送信することでアプリの過去全てのバージョンを収集した．つまりアプリの過去全てのバージョンを収集するというのは，アプリの最新版から一番古いバージョンまでの全てのバージョンの APK をダウンロードすることを意味する．上記のような値のデクリメントを繰り返し，ダウンロードのリクエストを送信するような手法を用いた時，versionCode の値が大きすぎると過去バージョンの収集に長い時間がかかってしまう．従って今回は既存研究 [3] を参考に過去バージョンを収集する対象のアプリを最新版の versionCode が 40,374 以下のアプリに限定し，アプリ収集の効率化を行った．

アプリをダウンロードする際に必要なデバイス ID は Nexus 5 のものを使用した．また， 1 つの Google アカウントにつきリクエスト毎に 5 秒のインターバルを挿入することで，一度もアカウントを停止されずにアプリを収集することができた．

### 2.3 脆弱性検知

収集したアプリに対して脆弱性検知を行った．

本研究では 1 章に述べたような理由からサードパーティライブラリベースの脆弱性に着目して調査を行った．従ってライブラリ由来の脆弱性を検知するために，APK に含まれているライブラリとそのバージョンを検知するツール，LibScout [3] を用いた．LibScout ではライブラリがどの程度 APK に含まれているのかを示す値 Similarity Score を算出するが，今回はこの値が 0.85 以上である場合のみそのライブラリが APK に含まれていると判断した．検知されたライブラリのバージョンが脆弱性が修正されたバージョンよりも古い場合は脆弱，そうでない場合は安全であると判断した．また， LibScout では検知したライブラリのバージョンの候補を複数提示する場合がある．本研究ではその際，一番新しいライブラリのバージョンを APK で実際に利用されているライブラリとして用いた．

App Security Improvement Program（以降， ASI Program ）はアプリ開発者に，よりセキュアなアプリを開発してもらうための知識などを提供する， Google が実施しているプログラムである．ASI Program ではいくつかの脆弱性について脆弱性修正キャンペーンを実施している．キャンペーンでは対象となった脆弱性を含んだアプリを公開している開発者に脆弱性を修正するよう通知を行い，修正の締め切りを設けている．ASI Program から開発者への通知は，自身の作成したアプリのリリースや管理を行う Google Play Console と，開発者へのメールを通して行われる．開発者が公開しているアプリにどのような脆弱性が存在していたのかが，その脆弱性を修正するためのサポートページへのリンクとともに通知される [4]．締め切りを過ぎると対象の脆弱性を含んだ新規アプリの公開が禁止されるだけではなく，既存アプリの脆弱性を含んだままのアップデートもブロックされ Google Play に公開できなくなる．本研究では ASI Program のキャンペーンで対象となっている脆弱性の内ライブラリベースの脆弱性に着目し，分析を行った．

### 2.4 脆弱性対応の分析

**図 2** にアプリに対して行った分析の概念図を示す．本論文ではアプリの脆弱性への対応を以下の 4 種類に分類する．

(1) Early Fix: キャンペーンの開始前に脆弱性を修正
(2) Campaign Fix: キャンペーンの期間中に脆弱性を修正
(3) Late Fix: キャンペーンの締め切り後に脆弱性を修正
(4) Unfixed: 脆弱性を修正していない

Early Fix ， Campaign Fix と Late Fix についてはアプリの最後の脆弱なバージョンから修正バージョンまでにかかった時間（ time-to-fix ）を計算した．Campaign Fix と Late Fix についてはキャンペーン開始日から修正までにか

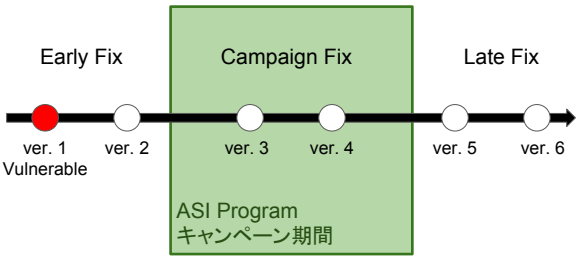


図 2　脆弱性対応の分析の概念図.

かった時間（ time-to-react ）を計算した．キャンペーン期間中に新規に公開された脆弱なアプリについては，新規公開から修正にかかったバージョンまでの時間を計算した．

時刻の取得には APK に必ず含まれている ”Android-Manifest.xml” のタイムスタンプを利用した．Android アプリを開発するための IDE である Android Studio ではアプリのビルドに Gradle を利用している．新しいバージョンの Gradle ではビルド時にデフォルトで APK に含まれるファイルのタイムスタンプが 1979 年 11 月 30 日に設定されるようになっている．このため今回収集した APK にもタイムスタンプが 1979 年 11 月 30 日 に設定された APK を確認できた．Gradle 以外の偽のタイムスタンプにも対応するために本研究では Google Play の前身である Android Market が一般ユーザ向けに公開された 2008 年 10 月 22 日より前のタイムスタンプは偽のタイムスタンプであると判断した．本研究では場合により偽のタイムスタンプを利用しているアプリや APK を除外して分析を行った．

## 3. データ

### 3.1 収集アプリ

日本とアメリカの Google Play における 34 種のカテゴリから最大でランキング上位 100 位のアプリを収集対象とした．さらにそれらのアプリの開発者が開発する他のアプリも収集対象に加えた．2017 年 7 月 23 日時点でのランキングと公開されているアプリを元に作成した収集対象は計 38,209 個のアプリとなった．アプリの収集は 2017 年 7 月 24 日から 2017 月 8 月 23 日にかけて行った．対象としたアプリの内，全ての過去バージョンを収集できたのは 21,046 個のアプリで計 142,611 個の APK をダウンロードした（最新版のみのアプリも含む）．全ての過去バージョンを収集できなかった原因としては，2.2 節で述べた versionCode の上限と，ネットワークのエラーなどがある．全てのバージョンを収集できたアプリの内，2 つ以上のバージョンがあったのは 12,790 個のアプリで，計 134,355 個の APK となった．よってアプリごとのバージョン数の平均は全てのアプリを用いると 6.78 ± 14.1（± 以降の数値は標準偏差を表す）であり，バージョンが 2 つ以上あるアプリのみを用いれば 10.5 ± 17.1 であった．

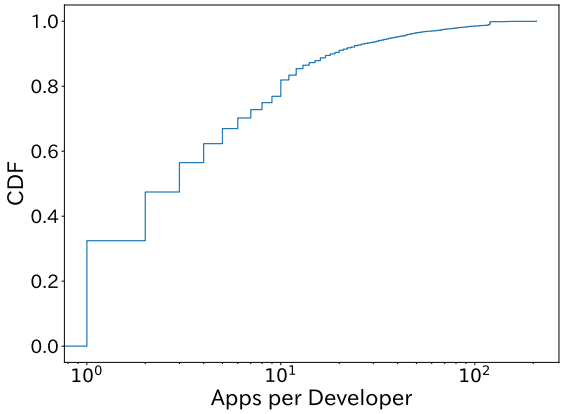


図 3　各開発者が公開しているアプリ数の累積分布.

アプリのメタデータ（開発者名，インストール数）の収集は 2017 年 8 月 14 日にアプリの収集とは別に行った．収集した 21,046 個のアプリの内，開発者名，インストール数の収集が失敗したのはそれぞれ 0.974 %，1.65 %であった．従ってメタデータを利用した分析では適宜収集が失敗したアプリを除外している．Google Play ではインストール数は 10,000,000〜50,000,000 のようにある一定の範囲で示される．本研究ではインストール数を用いた分析では下限の値を利用した．収集したアプリを開発した別々の開発者の数は合計 2,684 人となり，各開発者は平均で 7.76 ± 16.6 個のアプリを開発していた．またアプリを 2 つ以上公開している開発者に限れば，開発者の数は 1,732 人となり，各開発者は平均で 11.5 ± 19.7 個のアプリを開発していた．**図 3** に各開発者が公開しているアプリ数の累積分布を記す．

### 3.2 ライブラリと脆弱性検知

本研究では Android アプリに利用されているライブラリとそのバージョンを検知することができるツール LibScout を用いて脆弱性検知を行った．

ASI Program でキャンペーンとして取り上げられている脆弱性の内，調査対象とした脆弱性を**表 1** に示す．キャンペーンとなっているライブラリの内，多く検出されたものを選択した．LibScout を用いて調査対象とした 3 個のライブラリ Supersonic Ad SDK, MoPub Ad SDK, Vungle Ad SDK の利用の有無を調査した．

調査対象としているライブラリが検知されたアプリと APK の数，そして実際に収集できたアプリと APK の中での割合を**表 2** に記す．同様に本研究で対象としている脆弱なバージョンのライブラリが検知されたアプリと APK の数とその割合を**表 3** に記す．Vungle を利用したアプリの総数は Supersonic を利用しているアプリよりも多かった

表 1　ASI Program のキャンペーン.

| キャンペーン | 脆弱性の概要 | 期間 |
|---|---|---|
| Supersonic Ad SDK | バージョン 6.3.5 より前に存在する， Javascript により関数が漏洩する脆弱性 | 2016 年 9 月 28 日〜2017 年 1 月 26 日 |
| MoPub Ad SDK | バージョン 4.4.0 より前に存在する，ローカルリソース漏洩の脆弱性 | 2016 年 3 月 31 日〜2016 年 7 月 11 日 |
| Vungle Ad SDK | バージョン 3.3.0 より前に存在する，中間者攻撃につながる脆弱性 | 2015 年 6 月 29 日〜2015 年 11 月 11 日 |

表 2　ライブラリが検知されたアプリ・APK の数（収集したアプリ・APK 中での割合）.

| SDK 名 | アプリ数 | APK 数 |
|---|---|---|
| Supersonic Ad SDK | 300 (1.42 %) | 860 (0.603 %) |
| MoPub Ad SDK | 979 (4.65 %) | 4,560 (3.20 %) |
| Vungle Ad SDK | 316 (1.50 %) | 1,313 (0.921 %) |

表 3　脆弱なライブラリが検知されたアプリ・APK の数（収集したアプリ・APK 中での割合）.

| SDK 名 | アプリ数 | APK 数 |
|---|---|---|
| Supersonic Ad SDK | 69 (0.328 %) | 190 (0.902 %) |
| MoPub Ad SDK | 297 (1.41 %) | 937 (4.45 %) |
| Vungle Ad SDK | 0 (0.00 %) | 0 (0.00 %) |

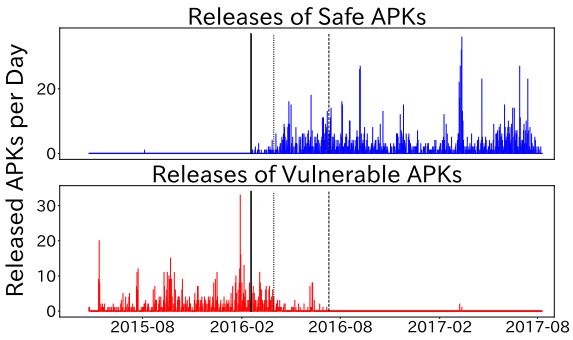


図 4　安全もしくは脆弱な MoPub を含んだ APK の日毎のリリース（三つの縦線は左から順に安全な MoPub のリリース，キャンペーンの開始，キャンペーンの締め切り）.

が，脆弱なバージョンを利用している APK を検知することはできなかった．よって 2.4 節で述べた分析は MoPub と Supersonic を利用しているアプリを対象に行う．

## 4. 分析結果

本章では本研究で対象としている脆弱性へのアプリの対応を 2.4 節で述べた手法を元に分析する．なお脆弱性対応への分類を行う際には，2008 年 10 月 22 日より前のタイムスタンプを用いて修正された 14 個（ 3.83%）のアプリは除外している．

### 4.1 キャンペーンへのマーケットの反応

図 4 に安全な MoPub を含んだ APK と，脆弱な MoPub を含んだ APK の Google Play への公開の様子を記す．安全なバージョンの MoPub がリリースされた後も脆弱なバージョンを利用した APK が公開されているのが確認できる．キャンペーンの開始とともに脆弱なバージョンを含

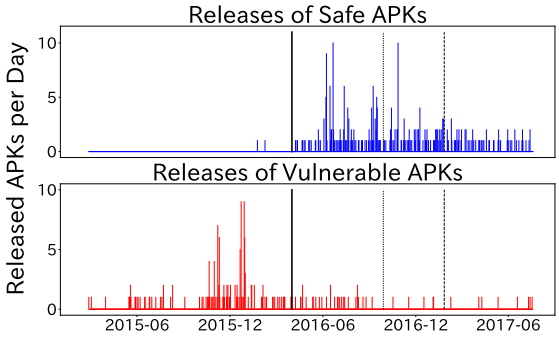


図 5　安全もしくは脆弱な Supersonic を含んだ APK の日毎のリリース（三つの縦線は左から順に安全な Supersonic のリリース，キャンペーンの開始，キャンペーンの締め切り）.

表 4　脆弱性対応の各種類の個数.

| | Supersonic Ad SDK | MoPub Ad SDK |
|---|---|---|
| Early Fix | 56(83.6%) | 35(12.3%) |
| Campaign Fix | 0(0.00%) | 136(47.7%) |
| Late Fix | 1(1.49%) | 73(25.6%) |
| Unfixed | 10 (14.9%) | 41(14.4%) |
| 合計 | 67 (100%) | 285 (100%) |

んだ APK のリリースが減少し，締め切りとともにほぼなくなることが見て取れる．キャンペーンが開始されると脆弱なバージョンを利用している開発者にはアップデートを促す通知が行われる．それにも関わらずキャンペーン開始後に脆弱なバージョンを利用した APK が 73 個リリースされているのが確認できた．

図 5 に安全もしくは脆弱な Supersonic を含んだ APK のリリースの様子を記す．安全なバージョンの Supersonic がリリースされると，キャンペーン開始前から安全なバージョンを利用した APK がリリースされている様子が確認できる．

今回行った分析ではまだリリースされていないはずの安全なバージョンのライブラリと，キャンペーンの締め切り後にも脆弱なバージョンのライブラリを含んだ APK のリリースが検知された．これの要因としては LibScout の誤検知やタイムスタンプの誤設定などが考えられる．

### 4.2 脆弱性ごとの比較

それぞれのアプリの脆弱性への対応を 2.4 節で述べた 4 種類に分類した．分類した結果を表 4 に記す．Supersonic では Early Fix が，一方 MoPub では Campaign Fix が一番多くなっている．これの一つの原因としてはライブラリ

表 5　脆弱性の修正・キャンペーンへの対応にかかった日数.

| | Supersonic Ad SDK | MoPub Ad SDK |
|---|---|---|
| time-to-fix | 58.0 ± 71.7 | 180 ± 147 |
| time-to-react | 165 ± 0.00 | 143 ± 137 |

の安全なバージョンのリリースからキャンペーンの開始までの時間が異なることが挙げられる．Supersonic は安全であるバージョン 6.3.5 のリリースから 180 日でキャンペーンが開始されたのに対し，MoPub は安全であるバージョン 4.4.0 のリリースから 42 日でキャンペーンが開始されている．Unfixed なアプリについてインストール数の和を計算することで合計のインストール回数を求めた．これにより脆弱な Supersonic と MoPub を今も修正していないアプリはそれぞれ最低でも 3,730 万回と 892 万回インストールされていることがわかり，今なおユーザに利用されている可能性がある．二つのキャンペーンを合計すると Early Fix, Campaign Fix, Late Fix, Unfixed なアプリの個数はそれぞれ 91(25.9%),136(38.6%),74(21.0%),51(14.5%) となった．また Unfixed なアプリは二つの脆弱性を合わせると，最低でも計 4,622 万回インストールされていることになる．

各脆弱性の time-to-fix と time-to-react の平均を**表 5** に記す．time-to-fix を見ると脆弱性への対応の分類と同様に Supersonic の脆弱性への対応が MoPub の脆弱性への対応よりも早く行われている．今回調査したアプリでは Supersonic の脆弱性について Campaign Fix と Late Fix だったアプリが 1 つのみだったので， time-to-react についての議論は今後の課題としたい．

### 4.3　アプリの人気

2.4 節で述べた Early Fix, Campaign Fix, Late Fix, Unfixed に属するアプリのインストール数の平均はそれぞれ 221 万回，640 万回，55 万回，91 万回となった．Late Fix と Unfixed の平均インストール数が少なくなっている．つまりインストール数が少ないアプリは ASI Program のキャンペーンに対応しにくいことが分かる．

**図 6** にアプリのインストール数と ASI Program のキャンペーンへの対応にかかった日数，time-to-react の関係を記す．この図からはインストール数と time-to-react の間に相関は見受けられない．よってどのようなアプリがキャンペーンに素早く対処し，脆弱性を修正しやすいのかは今後のさらなる調査対象である．

### 4.4　アプリ生成サービスにより生成されたアプリ

コーディングの必要のないアプリ生成サービスの一つに Mobincube [5] がある．Mobincube では GUI により，画像の配置やページの遷移などを指定することでコードを書くことなくアプリを生成することができる．アプリをマネタイズする方法として MoPub を利用した広告を追加する機能を提供しており，Mobincube により生成されたアプリには広告の利用に関わらず必ず MoPub Ad SDK が含まれている [6]．脆弱なバージョンのライブラリをアップデートするには，一度 Mobincube にてアプリを生成しなおしてから Google Play にアップロードすれば良い．

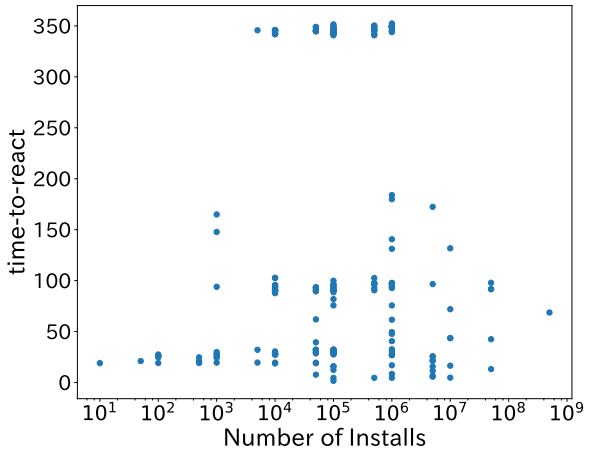


**図 6**　アプリのインストール数とキャンペーンへの time-to-react の関係.

Mobincube により生成されたアプリの割合は，収集できたアプリ全体の中では 0.442%だったが，MoPub のキャンペーンで Unfixed となったアプリの中では 63.4%と高くなっている．これは手軽にアプリを生成できる反面，長期的にアプリをメンテナンスする意思の弱い開発者が多くなってしまったためだと考える．

一方 MoPub が対象のキャンペーンへの time-to-react の平均は Mobincube により生成されたアプリの平均は 27.4 ± 4.44 日であり，それ以外のアプリの平均 180 ± 139 日よりも短くなっている．通常のコーディングを必要とするアプリ開発ではライブラリをアップデートする際には設定ファイルの編集などが必要となるが， Mobincube を利用した開発ではこの必要がない．このように脆弱なバージョンのライブラリをアップデートする負担が少ないことが，time-to-react の差につながっていると考えられる．

### 4.5　開発者の対応

ASI Program のキャンペーンの対象となった脆弱性を一度でも含むアプリを開発した開発者は，開発したアプリによって脆弱性への対応が異なるかを調査した．対象の脆弱性を含むアプリを 2 個以上開発した 72 人の開発者の内，アプリによって脆弱性への対応が異なったのは 15 人 ( 20.8% ) の開発者であった．これより脆弱性やキャンペーンへの対応は開発者ごとに似た傾向があることが確認できる．

各開発者が作成したアプリがそれぞれどのように脆弱性に対応したかによって求める開発者のスコア，$developerScore$

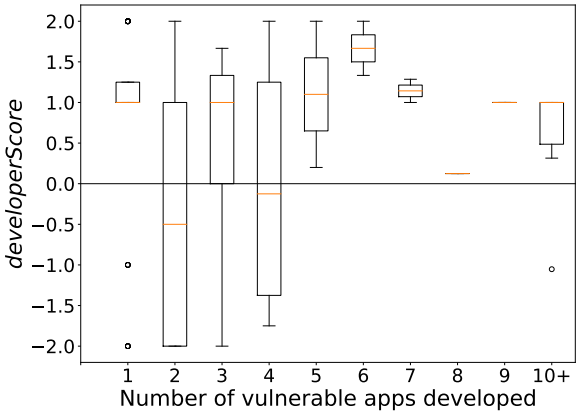


**図 7** 開発者が開発した脆弱なアプリ数と *developerScore* の関係.

を以下のように定義した.

$$\begin{aligned} developedAppsScore = {} & 2\,(\#\text{ of Early Fix apps}) \\ & +\,(\#\text{ of Campaign Fix apps}) \\ & -\,(\#\text{ of Late Fix apps}) \\ & -2\,(\#\text{ of Unfixed apps}) \end{aligned}$$

$$developerScore = \frac{developedAppsScore}{\#\text{ of vulnerable apps developed}}$$

*developerScore* が高い開発者ほどライブラリのアップデートやキャンペーンに敏感である．各開発者が開発した本研究での調査対象となっている脆弱性を含むアプリの数と，*developerScore* の関係を示した箱ひげ図を**図 7** に記す．箱の上下はそれぞれ第一四分位点，第三四分位点，丸記号は外れ値，箱の中の線は中央値，ひげは外れ値を除いた最大値，最小値を示す．脆弱性を含んだアプリを 10 個以上開発した開発者は一つの階級にまとめてある．8 個，9 個開発した開発者はどちらも一人しかいなかったため箱が潰れている．脆弱性を含んだアプリを 5 個以上作成した開発者たちのスコアが高い傾向にあること，そしてスコアがまとまっていることが確認できる．つまりキャンペーンの対象となった脆弱性を含んだアプリをより多く開発した開発者の方が，アプリのメンテナンス，脆弱性やキャンペーンなどへの対応が良い．本調査ではランキングに載っているアプリと，そのアプリと同じ開発者が開発した他のアプリを収集対象として加えた．しかし versionCode の上限などにより対象としたアプリ全てを収集することはできなかった．従って図 7 での開発者が作成した脆弱なアプリ数は正確な値ではないことに注意されたい．

## 5. 議論

本章では本研究での制約，今後の課題や脆弱性への対応についての提言を行う．

### 5.1 ASI Program で対象としている脆弱性

本研究では ASI Program での脆弱性修正キャンペーンの対象となっている脆弱性の内 JAR や AAR ファイルとしてアプリに組み込むことができるライブラリベースの脆弱性に着目した．キャンペーンの対象となっている脆弱性には開発者のコード由来やネイティブなライブラリ由来の脆弱性などもある．脆弱性の由来によってアプリの対応が異なるかどうかの調査は今後の課題である．

### 5.2 Supersonic と MoPub の対応の差

4.2 節では Supersonic と MoPub の脆弱性への対応に差があったことを述べた．これにはライブラリアップデート時に開発者にかかる負担が関わっていると考えられる．しかし今回は当時のドキュメントを確認することができなかったため，ライブラリアップデートの負担を調査する手法の考案が必要である．

また，脆弱性の深刻度も関係していると考える．分析対象とする脆弱性を増やした上で，深刻度による対応の違いの調査も今後の課題である．

### 5.3 分析対象のサンプル

今回はキャンペーンの対象となった脆弱性を含むアプリ，合計 352 個を主な分析対象として用いた．より多くのサンプルを調査することができれば，より正確な結果が得られる．また，収集対象としてランキング上位のアプリを中心とし，アプリを実際に収集する際に versionCode の最大値も設けたためサンプルに偏りが生じている可能性もある．アプリの収集対象とサンプル数を増やし，再び調査することは今後の重大な課題である．

定常的にアプリを収集することも課題である．これによりアプリのバージョンの正確なリリース日を知ることができ，versionCode の最大値によるサンプルの偏りも緩和される．

### 5.4 脆弱性への対応

本研究では ASI Program のキャンペーンで対象となった脆弱性を修正していないアプリが今も Google Play で公開されていることを示した．脆弱性が未修正なアプリの詳細ページを見ても脆弱性に関する記述はされていない．キャンペーンの対象にするような脆弱性ならば，脆弱性のマーケットからの完全な駆除，もしくはユーザに危険性を知らせる何らかのチャネルを設けるべきである．

アプリ生成サービスによって作成されたアプリは脆弱性が修正されにくいことを示した．このようなサービスを利用しているアプリのメンテナンスを行いにくい開発者に対してライブラリアップデートや脆弱性対応を効率良く周知し，実施してもらうシステムを模索すべきである．

## 6. 関連研究

### 6.1 複数のバージョンを利用した調査

アプリの複数のバージョンをダウンロードするには過去のバージョンをリクエストするのではなく，定期的にマーケットをクロールする方法もある．Viennot ら [7] は定期的に Google Play 上のアプリとそのメタデータを収集するクローラ， PlayDrone を作成した．

Taylor ら [8] は，同じアプリの古いバージョンと新しいバージョンを比較し要求するパーミッションの変化を調査した．この結果， Google Play で配布されているアプリは時間の経過とともにより多くのパーミッションを要求するようになっていることを報告した．同様に脆弱性の増減についても調査を行った．これにより，人気なアプリはランダムに抽出されたアプリと比較し，より脆弱になっていることを明らかにした．また，サードパーティライブラリが潜在的にアプリに脆弱性を組み込む危険性を報告している．

### 6.2 Android アプリの脆弱性検知

Fahl ら [9] は，Android アプリの SSL/TLS の誤った使い方を検知するツール MalloDroid を作成した．具体的にはアプリの，全ての SSL 証明書を許可する実装や，証明書で指定されているドメインを無視して証明書を許可するなどの実装をアプリコードの静的解析により発見する．MalloDroid を用い人気アプリの中にも不適切な SSL/TLS の使用があることを検知した．検知したアプリの一部に対し MITM 攻撃を行い実際に通信が盗聴できることを報告した．

Backes ら [3] は，Android アプリに使われているライブラリとそのバージョンを検知するツール LibScout を作成した．クラスタリングを用いたライブラリ検知とは異なり，検知したライブラリのバージョンも識別することができる．このため，使用しているライブラリが脆弱なバージョンなのかそうでないのかも判定することができる．脆弱な Facebook SDK と Dropbox SDK へのマーケットの反応を調査し，安全なバージョンがリリースされてからも脆弱なバージョンを利用したアプリが多数マーケットでリリースされていることを明らかにした．

## 7. まとめ

Google Play のランキング上位のアプリを中心に収集対象を決定した．収集対象のアプリについて過去全てのバージョンを収集し，ASI Program の一環として実施される脆弱性修正キャンペーンで対象の脆弱性へのアプリの対応を調査した．アプリの脆弱性への対応を Early Fix, Campaign Fix, Late Fix, Unfixed の 4 種類へ分類した．これにより今回収集した，調査対象の脆弱性を含んだアプリの内，14.5%が今も修正を行わないまま配信されていることを明らかにした．このような脆弱性が未修正なアプリは最低でも計 4,622 万回インストールされている．MoPub の脆弱性が修正されていないアプリの内 63.4% がコーディングを必要としないアプリ生成サービス（ Mobincube ）によるものだと明らかにした．一方でサービスより生成されたアプリの MoPub が対象のキャンペーンへの対応にかかった時間の平均は 27.4 日と，その他のアプリの平均 180 日よりも素早く対応していることが分かった．これは設定ファイルの変更などが必要ないアップデートの簡便さによるものだと考えられる．またダウンロード数の少ないアプリはキャンペーンへの対応が良くないことを確認した．今後は収集するアプリと，調査を行うキャンペーンを増やし，より正確な結果を得ることが重大な課題である．

**謝辞** 本研究の一部は JSPS 科研費 16H02832 の助成を受けたものです．

### 参考文献

[1] AppBrain: Google Play stats, `http://www.appbrain.com/stats/`.
[2] Watanabe, T., Akiyama, M., Kanei, F., Shioji, E., Takata, Y., Sun, B., Ishi, Y., Shibahara, T., Yagi, T. and Mori, T.: Understanding the Origins of Mobile App Vulnerabilities: A Large-scale Measurement Study of Free and Paid Apps, *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pp. 14–24 (2017).
[3] Backes, M., Bugiel, S. and Derr, E.: Reliable Third-Party Library Detection in Android and Its Security Applications, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, New York, NY, USA, ACM, pp. 356–367 (online), DOI: 10.1145/2976749.2978333 (2016).
[4] Google Inc.: App Security Improvement Program, `https://developer.android.com/google/play/asi.html`.
[5] Mobincube: Mobincube, `https://www.mobincube.com/`.
[6] Mobincube: Mopub's SDK issue fixed, `http://blog.mobincube.com/mopubs-sdk-issue-fixed/` (2016).
[7] Viennot, N., Garcia, E. and Nieh, J.: A Measurement Study of Google Play, *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, New York, NY, USA, ACM, pp. 221–233 (online), DOI: 10.1145/2591971.2592003 (2014).
[8] Taylor, V. F. and Martinovic, I.: To Update or Not to Update: Insights From a Two-Year Study of Android App Evolution, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, New York, NY, USA, ACM, pp. 45–57 (online), DOI: 10.1145/3052973.3052990 (2017).
[9] Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B. and Smith, M.: Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security, *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, New York, NY, USA, ACM, pp. 50–61 (online), DOI: 10.1145/2382196.2382205 (2012).