署名情報を利用した Android マルウェアの推定手法の提案

西田雅太† 神薗雅紀† 星澤裕二†

†株式会社セキュアブレイン 先端技術研究所
〒102-0083 東京都千代田区麹町 2-6-7 麹町 RK ビル 4F
E-mail: †{ masata_nishida, masaki_kamizono, yuji_hoshizawa}@securebrain.co.jp

あらまし Android アプリケーションを配布するためには署名が必要となり、署名には通常「自己署名証明書」が用いられる。これゆえ、アプリケーションの署名情報を確認することで、同一の証明書を使って署名されたものであるか否か容易に判定することが可能である。これを利用し、既知の Android マルウェアの証明書情報を蓄積することによって、ヒューリスティックに Android マルウェアを検知する要素技術のひとつとすることを提案する。また、提案手法の検証として、複数の Android マルウェアから署名情報を抽出し、提案手法の有効性や同一証明書の使用状況について調査する。

Android Malware Heuristics using Digital Certificates

Masata Nishida† Masaki Kamizono† and Yuji Hoshizawa†

†Advanced Research Laboratory, SecureBrain Corporation
Kojimachi RK Bldg., 6-7 Kojimachi 2-chome, Chiyodaku, Tokyo, Japan
E-mail: †{ masata_nishida, masaki_kamizono, yuji_hoshizawa}@securebrain.co.jp

Abstract An Android application must be digitally signed to be used on a device, and usually a self-signed certificate is used by a developer. Based on the signature, we can easily check if other apps are digitally signed using the same certificate. This paper will propose the use of digital certificate information of well-known Android malware to detect other potential variants heuristically. This paper will also evaluate the proposed method by using a sampling of various Android malware.

1 はじめに

近年、Android 端末が急速に普及している。それに伴い、Android プラットフォームをターゲットにしたマルウェアも急増している。その中で、単純な方法で検知することが困難なポリモーフィック型マルウェアも出現し、Android 端末においても適切なマルウェア対策によるユーザの安全性の確保が重要な課題となっている。

そこで本稿では、Android アプリケーションにおけるデジタル署名に着目し、マルウェアの推定を行う手法を提案する。提案手法では、同一の証明書を用いて署名されたアプリケーションは、同一の開発者によって開発されたものであるという仮定のもとに、既知のマルウェアと同一の証明書を用いて署名されているアプリケーションをマルウェアと推定する。

提案手法について述べたのちに、その妥当性について検証を行う。検証では、著者らが独自に収集した約 15,000 の Android マルウェアにおいて、どの程度同一の証明書が署名に使用されているかについて調査する。また、10,000 の一般の Android アプリケーションに対して、既知のマルウェアの証明書が署名に使用されているかを調査し、提案手法を用いた際に一般アプリケーションをマルウェアと判定してしまう誤認率について検証を実施する。

検証に使用したデータセットの詳細については、4.1 で説明する。

2 Android の署名機構

2.1 Android における署名の役割

Android では、アプリケーションの配布に際し、アプリケーションに対してデジタル署名を行う必要がある。

Windows 環境において、アプリケーションに対するデジタル署名は、アプリケーションの開発者の身元を証明するために行われる。一方、Android では第三者認証局から発行された証明書を使用することも可能ではあるが、もっぱら自己署名証明書によって署名されることが多い。これは、次に述べるような Android におけるデジタル署名の目的・用途に起因するものである。

Android ではアプリケーションの更新を行う際に、既にインストールされている旧バージョンと新バージョンのアプリケーションとが同じ証明書で署名されていないと、更新が行えない。また、同一の証明書を使って署名されたアプリケーション間でリソースの共有を行うことが出来る。

このように、Android アプリケーションの署名情報は、アプリケーションの開発者を特定するためのものではなく、複数のアプリケーション間で開発者が同一であることを確認するために使用される。[1][2]

なお、前述のとおり Android アプリケーションは全てデジタル署名が施されているが、Android OS として端末上で署名情報を確認する機能はない。このことも、Android におけるデジタル署名がアプリケーション開発者の確認を目的としていないことひとつの証左といえる。

2.2 署名情報の抽出

Android アプリケーションの構造について説明したのち、証明書情報の抽出方法を述べる。

2.2.1 Android アプリの構造

Android アプリケーションは、apk という形式で配布される。apk ファイルの実体は zip 形式のアーカイブファイルである。

apk ファイルを展開した際のディレクトリ構造の例を以下に示す。

```
sample.apk/
├── AndroidManifest.xml
├── META-INF/
│   ├── ANDROID.RSA
│   ├── ANDROID.SF
│   └── MANIFEST.MF
├── assets/
│   ├── rageagainstthecage
│   └── sqlite.db
├── classes.dex
├── lib/
│   └── armeabi/
│       └── libandroidterm.so
├── res/
│   ├── drawable/
│   │   ├── icon.png
│   │   └── minispiral.png
│   └── layout/
│       └── main.xml
└── resources.arsc
```

apk ファイルに含まれるファイルのうち、主だったものについて説明する。

AndroidManifest.xml はアプリケーションのコンポーネント構成や使用するリソースを記述したファイルである。classes.dex は、プログラム実行コードであり、Android の Java VM(Dalvik)によって実行される。そして、署名に関係した情報は、META-INF ディレクトリ内に格納されている。

2.2.2 署名情報

META-INF ディレクトリは次の 3 つのファイルから構成される。

- MANIFEST.MF
- CERT.SF
- CERT.RSA

これらのファイルは、Android の SDK に付属する署名ツールによって生成することができる。署名ツールのソースコードは Android OS のソースコードとともに

公開されている(build/tools/signapk/SignApk.java)ため、ここでは概要のみ説明する。

MANIFEST.MF

MANIFEST.MF は apk ファイルに含まれる各ファイルの SHA1 ハッシュ値を Base64 エンコードしたものが記載されている。以下に MANIFEST.MF の冒頭部を示す。

```
$ cat META-INF/MANIFEST.MF
Manifest-Version: 1.0
Created-By: 1.0 (Android)

Name: res/layout/main.xml
SHA1-Digest: DIcN0dUH+Y0GNayVSLP9K0HaNco=
```

CERT.SF

CERT.SF は、前述の MANIFEST.MF のファイルの SHA1 ハッシュ値と、各エントリの SHA1 ダイジェスト値をとったものである。以下に、CERT.SF の冒頭部を示す。

```
$ cat META-INF/CERT.SF
Signature-Version: 1.0
Created-By: 1.0 (Android)
SHA1-Digest-Manifest: weQA9ZWOYI7sGMOfIQ7AF8zj+zg=

Name: res/layout/apk_item.xml
SHA1-Digest: C40fsazt9sCksdhZBq5hVee0/yA=
```

CERT.RSA

CERT.RSA は、CERT.SF に対し、RSA 暗号鍵と X.509 証明書で署名し、PKCS7 形式で表現したものである。従って CERT.RSA には、署名に使用した証明書のデータが含まれており、次節で示すように証明書データを抽出することが出来る。本稿の提案手法では、この CERT.RSA から抽出した証明書を利用し Android マルウェアの推定を行う。

2.2.3 証明書の抽出

以下のように、OpenSSL コマンドを用いて、前述の.RSA ファイルから証明書情報を取得することが出来る。

```
$ openssl pkcs7 -inform DER -in META-INF/CERT.RSA -n
oout -print_certs –text
```

また、スクリプト言語などを用いて証明書情報を抽出し、プログラム的に処理することもできる。

本稿の検証では、Ruby を用いて証明書情報の抽出を行った。Ruby に組み込まれている OpenSSL ライブラリを使用して、証明書情報を抽出し、ハッシュ値をとって比較に用いた。証明書抽出とハッシュ値化のサンプルコードを以下に示す。

```
require ‘openssl’
require ‘digest/sha1’

# read file
data = File.open('META-INF/CERT.RSA', 'rb').read
pkcs = OpenSSL::PKCS7.new(data)

# ‘cert’ is OpenSSL::X509::Certificate object
cert = pkcs.certificates[0]

# calculate hash digest
puts Digest::SHA1.hexdigest(cert.to_der)
```

3 マルウェア推定手法の提案

3.1 証明書と開発者の同一性

2.1 で示した通り、署名情報は Android OS においてはアプリケーション開発者の同一性を確認するために用いられている。

デジタル署名には、証明書と RSA 暗号鍵が必要であるが、Android では自己署名証明書による署名も許可しており、証明書の入手・作成は容易である。

以上のことから、同一の証明書で署名されたアプリケーションは、同一の開発者が署名したものであると推測することができる。

また、現在見つかっているポリモーフィック型 Android マルウェアは、OpFake[3]のように機械的に生成されていると考えられる。これらのマルウェアは、apk ファイル内にランダムで複数個の不要なファイルを差し込むなどの手法で、ファイルのハッシュ値を変化させている。このように比較的単純な手法でポリモーフィックを実現している点を考慮すると、アプリケーション生成時に毎回新しい証明書を作成して署名に用いているとは考えにくい。

そのため、ポリモーフィック型 Android マルウェアにおいても、同一種のマルウェアは同一の証明書を使っている可能性が高いと推測できる。

3.2 既知マルウェアを利用した推定

既知のマルウェアで署名に使用されている証明書と同一の証明書を使って署名されたアプリケーションは、前述のとおり当該マルウェアの作成者が署名し

たものと推測できる。よってそのアプリケーションもマルウェアである可能性が高いといえる。

そこで本稿では、まず既知のマルウェアの証明書情報を蓄積してデータベース化し、次にマルウェア判定を行いたいアプリケーションの証明書がデータベースの中に存在する場合、マルウェアと推定する手法を提案する。

提案手法の概略図を図 1 に示す。

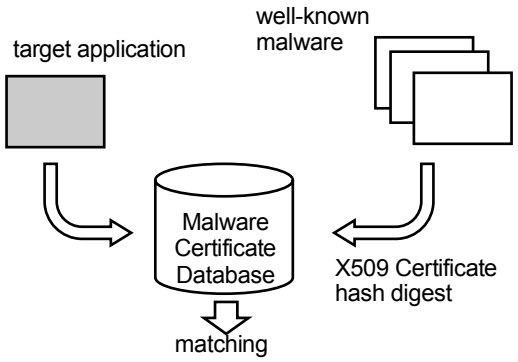

図 1 既知のマルウェアを用いた判定

既知のマルウェアの証明書情報(以下、既知マルウェア証明書)を利用することで、既知のマルウェアの開発者が作成した未知のマルウェアを検知できることが期待できる。

4 検証

提案手法の妥当性、有効性を確認するために以下の 3 つの検証を行う。

- 既知マルウェア証明書の重複使用
- 証明書の使用期間の調査
- 一般アプリケーションとの比較

まず、本手法の有効性を検証するために既知マルウェア証明書がどの程度重複しているかについて調査する。

次に、既知マルウェア証明書のうち、複数のマルウェアで署名に用いられている証明書がどの程度の期間、使用されているかについて調査する。

最後に、独自にアプリ配布サイトから収集したアプリケーションにおいて、どの程度既知マルウェア証明書が使用されているかを調査し、提案手法の誤認率について検証する。

4.1 検証データ

4.1.1 検証用マルウェア検体

提案手法を検証するためのマルウェアデータセットとして、著者らが独自に収集し SecureBrain アンチウイルスβ版[4]で Android マルウェアと判定された14717検体を使用する。これらの検体は SecureBrain アンチウイルスβ版で 136 ファミリーに分類された。

マルウェアのファミリー分布を表 1 に示す。

表 1 マルウェアファミリー分布

ファミリー	検体数
AndroidOS.FakeInst	4911
AndroidOS.Kmin	2464
AndroidOS.OpFake	2360
AndroidOS.Boxer	1399
AndroidOS.DroidKungFu	824
AndroidOS.Lotoor	432
AndroidOS.GingerMaster	272
AndroidOS.SmsSend	221
AndroidOS.SmsAgent	209
AndroidOS.JiFake	137
その他	1488
計	14717

検証用マルウェアデータは、FakeInst や OpFake などポリモーフィック型のマルウェアが占める割合が多くなっている。

事前に除外した検体

Android の OS のソースコードには、Android プラットフォームに含まれるアドレス帳や電話アプリなどの基本的なアプリケーションを署名するための、開発者用証明書と秘密鍵が付属している。

これらの証明書を使用して署名されたマルウェアも存在する。一方でこれらの証明書を使用した非マルウェアも多く存在するため、証明書のみを判断材料とする提案手法では、これらの証明書で署名されたアプリケーションの判定は行えない。

以上のことを踏まえて、これらの証明書を用いたマルウェアは事前に除外した。前節で述べた 14,717 検体は、これらの証明書を事前に除外してあるデータセットである。

事前に除外した検体は以下の 1022 個あった。その内訳を表 2 に示す。

表 2 事前に除外した検体

開発者用証明書	検体数
testkey.x509.pem	1001
media.x509.pem	18
shared.x509.pem	2
platform.x509.pem	1
計	1022

4.1.2 一般アプリケーション

FP 率などの検証のため、著者らが Android アプリケーション配布サイトから独自に収集した 10000 個の Android アプリケーションを使用する。アプリケーションの収集は、2012 年 5 月 14 日から 2012 年 7 月 2 日にわたって実施した。

収集したアプリケーションは、マルウェア対策ソフト等による検証は行なっていない。そのため、このデータ内にマルウェアが存在している可能性は否定出来ない。ただし、4.4 のマルウェアと一般アプリの比較結果からもわかるとおり、無視できる範囲といえる。

これらのアプリケーションから証明書を抽出したところ、ユニークな証明書は 5560 個存在していた。

4.2 同一証明書を使用したマルウェア

4.2.1 マルウェアデータセット全体の重複

マルウェアデータセット 14717 個の検体から、署名データを抽出したところ、証明書は 589 種類に集約された。

その内訳を表 3、図 2 に示す。

表 3 証明書重複使用状況

	証明書数	検体数	%
重複なし	317	317	2.15%
2～10	221	826	5.61%
11～50	37	984	6.69%
51～100	4	284	1.93%
101～500	5	1054	7.16%
501～1000	2	1390	9.44%
1001 以上	3	9862	67.01%
計	589	14717	

マルウェアデータセット全体で、証明書が 1 回しか使用されていないものは 317 検体で、全体の約 2%程度であり、残りのほとんどの検体において、複数のマルウェアの署名に同じ証明書が使用されている。

また、署名に使用された検体の数が 1000 を超える証明書が 3 つ存在し、マルウェアデータセット検体数の 6 割以上を占めているが、これらのマルウェアはほぼポリモーフィック型であった。

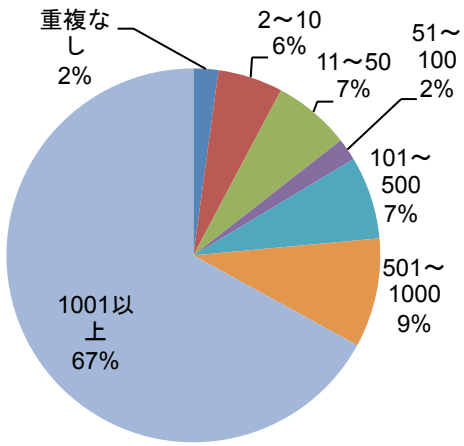

図 2 証明書重複検体数

4.2.2 主なポリモーフィック型の除外

マルウェアデータセットの検体はポリモーフィック型マルウェアの占める割合が多い。そのため、4.2.1 の検証では、ポリモーフィック型マルウェアによる影響が大きい。

そこで、FakeInst, OpFake, Boxer, Kmin の 4 つの主だったポリモーフィック型マルウェアのファミリーを除外して、非ポリモーフィック型における提案手法の有効性の検証を行う。4 種のファミリーを除外した証明書の重複集計結果を、表 4、図 3 に示す。

表 4 ポリモーフィック型を除外した証明書重複状況

	証明書数	検体数	%
重複なし	315	315	8.79%
2～10	211	778	21.71%
11～50	35	893	24.92%
51～100	6	486	13.56%
101～500	4	1111	31.01%
	571	3583	

主だったポリモーフィック型を除外してもなお、90%以上の検体で証明書が使いまわされていることが分かる。

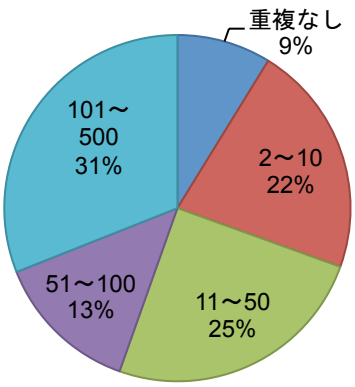

図 3 ポリモーフィック型を除外した証明書重複検体数

4.2.3 ポリモーフィック型の証明書

前節で除外したポリモーフィック型のマルウェアの署名に用いられた証明書についても調査を行った。

OpFake、FakeInst、Boxer、Kmin の 4 種のポリモーフィック型マルウェアで、署名に使用された証明書数についてまとめた結果を表 6 に示す。

表 5 ポリモーフィック型の証明書分布

	検体数	証明書
OpFake	2360	9
FakeInst	4911	31
Boxer	1399	4
Kmin	2464	2

FakeInst 以外の 3 つのファミリーでは、利用証明書数は 1 桁であり、非常に少ない種類の証明書で署名されていることが分かる。

また、各ファミリーで一番多く使用されていた証明書で署名されていた検体数を表 6 に示す。

表 6 ポリモーフィック型の最大署名数

	検体数	最大署名数	%
OpFake	2360	2288	96.9%
FakeInst	4911	2602	53.0%
Boxer	1399	1376	98.4%
Kmin	2464	2447	99.3%

FakeInst 以外の 3 つファミリーでは、95%以上の検体が 1 つの証明書で署名されている。FakeInst は証明書の数も多いが、それでも 1 つの証明書で 50%以上の検体が署名されている。なお、FakeInst と Boxer の最大重複の証明書は同一の証明書である。

4.3 証明書の使用期間

複数のマルウェアの署名に使われている証明書が、どの程度の期間使用されているのかについて調査を行った。本稿では apk(zip) ファイル内の AndroidManifest.xml エントリの日時をアプリケーションの作成日時とし、同一証明書を用いたアプリケーションの中で、最も古い作成日時と最も新しい作成日時の差を証明書使用期間と定義した。

4.3.1 証明書使用期間の分布

マルウェアデータセット内の証明書 589 個のうち、1 つのマルウェアでしか署名に使用されていない、317 個を除いた 272 個の証明書の使用期間を表 7、図 5 に示す。

表 7 証明書の使用期間

	証明書		検体数	
1 日未満	44	16.18%	171	1.19%
1 ヶ月未満	83	30.51%	432	3.00%
1 ヶ月以上 2 ヶ月未満	44	16.18%	187	1.30%
2 ヶ月以上 3 ヶ月未満	14	5.15%	178	1.24%
3 ヶ月以上 半年未満	36	13.24%	792	5.50%
半年以上 1 年未満	38	13.97%	9876	68.58%
1 年以上	13	4.78%	2764	19.19%
	272	100.00%	14400	100.00%

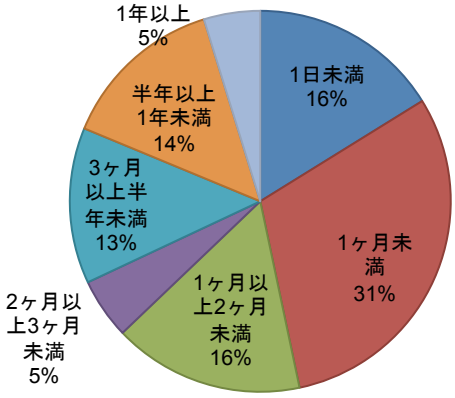

図 4 証明書の使用期間

50%以上の証明書が、1 ヶ月以上の期間にわたって使用されていた。また、1 年以上の長期にわたって

使用されている証明書も13個確認できた。

4.3.2 マルウェア作成日の分布

使用期間の長い証明書が、その期間内でどのように使用されていたかをみるために、最も多くの検体に対して用いられた証明書に着目し、その証明書で署名されたマルウェアの作成日の分布を調査した。

対象の証明書は、FakeInst などのポリモーフィック型マルウェアで使用されていたもので、4078 個のマルウェアが署名され、2011/8/1 から 2012/7/13 までの約 1 年弱の期間利用されていた。この証明書を使ったマルウェア作成日の分布を図 5 に示す。

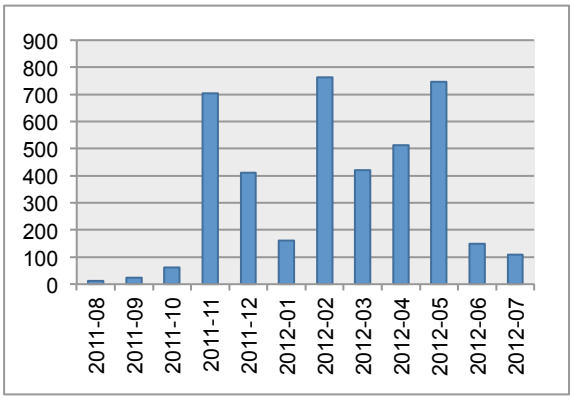

図 5 マルウェア作成日の分布

対象の証明書を使用したアプリケーションの作成日は、2011年8月に使用開始されてから、2011年11月以降に急激に増えている。その後は多少の増減はあるものの継続的に使用されていることがわかる。

4.4 一般アプリとの比較

提案手法の誤認率を検証するため、マルウェアと一般アプリケーションとの間で使用されている証明書にどの程度重複があるか調査を行った。

4.2.1 のマルウェアの証明書 589 個と、4.1.2 で示した一般アプリケーションの証明書 5560 個の重複状況をまとめた結果を表 8 に示す。

表 8 一般アプリとの証明書の重複

証明書ダイジェスト	一般アプリ数	マルウェア数
813a3a…	40	35
bc87c8…	21	2
8ef766…	16	1
4867c0…	10	1
a7ebbf…	4	7
5114e0…	4	2
24bb24…	4	2
2dce4f…	2	4
51b06b…	2	3
bef323…	2	1
e4130a…	2	2
b7d9c6…	1	3
20b959…	1	1
02dc79…	1	6
294337…	1	1
6230b6…	1	1
a0df7b…	1	2
63f9c6…	1	1
495857…	1	1
計	115	76

証明書の重複は全部で 19 個存在し、アプリ数としては、115 個の一般アプリケーションがマルウェア検体と同じ証明書を使用していた。

このうち、アプリケーション数が一番多かった証明書(813a3a…)のマルウェアは、SecureBrain アンチウイルスβ版ではフェイクアプリと分類された。一方で、同証明書で署名された一般アプリケーションの一部を VirusTotal[5]で検証したところ、複数のアンチウイルスソフトがフェイクアプリと判定していた。

4.1.2 の一般アプリケーションを、すべてマルウェアではないと仮定した場合の FP 率を表 9 に示す。

表 9 誤認率

	全体数	重複数	誤認率
証明書	5560	19	0.34%
アプリ	10000	115	1.15%

証明書数ベースでの FP 率は 0.5%未満であった。一方、アプリケーション数ベースで考えた場合は、1%以上になるものの、これは表 8 に示した通り、重複した証明書の中で一般アプリケーションに対して複数の署名に用いられているものが存在するためである。

5 考察

5.1 提案手法の妥当性

4.2.1 の結果より、約 15000 のマルウェアが 600 弱

の証明書で署名されていた。また、1つのマルウェアでのみ使用された証明書が約 300 と全体の 2%程度であり、非常に多くのマルウェアで同一の証明書を使って署名をしている傾向があることが分かった。また、検証マルウェアデータの多くを占めていたポリモーフィック型の影響を排除してもなお 90%以上のマルウェアで証明書の使い回しが見られたことから、一般的な Android マルウェアに対して提案手法が有効であるといえる。

一方、ポリモーフィック型 Android マルウェアに関しては、4.2.3 で示した通り、同種のポリモーフィック型マルウェアでは同一の証明書が利用される傾向があり、提案手法は現状のポリモーフィック型マルウェアに対しても非常に有効である。

次に、4.3 の検証により、長期にわたって使用され続ける証明書の存在を示した。これは、マルウェアの証明書情報を蓄積して用いる提案手法が、データ蓄積以降に作成されたマルウェアに対しても推定が行える可能性を示している。

4.4 で行った一般アプリケーションとマルウェアの証明書の比較により、多少の FP は確認できたものの概ね良好な結果が得られた。

以上のことから、提案手法は、証明書情報のみでマルウェアを推定するというごく単純な手法にもかかわらず、現時点において非常に有効な手法であるといえる。

5.2 課題・検討事項

5.2.1 情報蓄積と推定精度

提案手法は、既知のマルウェアの証明書情報を蓄積することでマルウェアの推定を行うため、本手法を使用するためには事前に既知のマルウェアの情報を収集する必要がある。そして、蓄積情報が多ければ多いほど、推定できるマルウェアが増えることから、いかに多くの Android マルウェアを集められるかが、提案手法の精度を上げる上での課題となる。

5.2.2 網羅性

提案手法は、既存のマルウェア情報を基にした推定であるため、世の中に存在するマルウェア全体を網羅して推定できる手法ではない。本手法ではまったくの未知のマルウェアや、新たな証明書を使用したマルウェアを推定することはできない。

5.2.3 ポリモーフィック型と証明書生成

ポリモーフィック型 Android マルウェアに関して、今回の検証では非常に良い結果が得られたが、マルウェア生成のアルゴリズムに証明書の生成も組み込まれた場合には、提案手法を利用することができない。

5.2.4 例外となる証明書

4.1.1 で述べたように、Android OS のソースコードに付属している開発用証明書と秘密鍵を使用したマルウェアサンプルは今回の検証では除外した。

開発者用証明書を使用したマルウェアが現状でも一定数存在し、提案手法ではこれらのマルウェア推定は行えない。また、今後これらの証明書を使用したマルウェアが増加した場合、提案手法の有効性が低下する。

5.3 今後の発展

本稿では、証明書情報のみを使用してマルウェアの推定を行った。今後はアプリケーションのマニフェスト情報や実行コードの情報など、他の要因も組み合わせることによって、精度の向上や 5.2 で示した問題点の改善に繋げることができると思われる。

6 おわりに

本稿では、署名情報を利用した Android マルウェアのヒューリスティックな推定手法を提案し、実際のマルウェアデータセットを使用した検証においてその有効性を示した。今後は、5.3 で述べたような推定精度の向上をはかりたい。

参考文献

[1] Android Security Overview | Android Open Source, http://source.android.com/tech/security/index.html#application-signing

[2] Signing Your Application | Android Developers http://developer.android.com/tools/publishing/app-signing.html

[3] Server-side Polymorphic Android Applications | Symantec Connect Community, http://www.symantec.com/connect/blogs/server-side-polymorphic-android-applications

[4] SecureBrain アンチウイルスβ版 – Google Play https://play.google.com/store/apps/details?id=jp.co.securebrain.Antivirus

[5] VirusTotal, https://www.virustotal.com/