

制御フロー解析による Android マルウェア検出方法の提案

岩本 一樹†

和崎 克己‡

† 日本コンピュータセキュリティリサーチ株式会社
422-8052 静岡県静岡市駿河区緑が丘町 1-1
iwm@maid.org

‡ 信州大学大学院総合工学系研究科
380-8553 長野県長野市若里 4-17-1
wasaki@cs.shinshu-u.ac.jp

あらまし Android におけるマルウェアは昨年末から本年にかけて増加しており、その対策が求められている。本研究では制御フロー解析を用いた Android で動作するマルウェアの検出方法を提案する。初めに技術者によって解析済みの Android のマルウェア検体を制御フロー解析してその特徴をパターンとして抽出した。大量にある未解析の Android アプリケーションを同様に制御フロー解析し、抽出したパターンが未解析の Android アプリケーションに含まれているかを調べることでマルウェアの検出を試みた。結果、未解析の Android アプリケーションの中からいくつかのマルウェアを検出することができた。

Detection Method for Android Malware by using Control Flow Analysis

Kazuki Iwamoto†

Katsumi Wasaki‡

†Japan Computer Security Research Center
1-1 Midorigaoka-cho, Suruga-ku, Shizuoka-shi, 422-8052 JAPAN
iwm@maid.org

‡Interdisciplinary Graduate School of Science and Technology, Shinshu University
4-17-1 Wakazato, Nagano-shi, 380-8553 JAPAN
wasaki@cs.shinshu-u.ac.jp

Abstract Android malware has been an increase from the end of last year, so measure are required. In this paper, we propose the method to detect Android malware by using control flow analysis. First off, we got control flows from Android malware samples which are already analysed by engineers, and extracted features as pattern. Secondly, we got control flows from huge amount of unanalysed Android applications, and we detected malware by looking at how an application includes extracted patterns. Finally, we could detect same malware samples in unanalysed Android applications.

1 はじめに

近年、スマートフォンは急速に普及しており、特に日本国内では Android が最も多くのシェアを獲得している [1]。それに伴い Android におけるマルウェアは増加しており [2]、その対策が

求められている。

Android ではアプリケーションが必要とする権限をマニフェストに記述し、アプリケーションの導入時にユーザが承認する仕組みがある。しかしユーザがマニフェストを見て判断することは難しい。そこで PC のようにマルウェアを

検出するソフトウェアが必要とされており、実際に Android Market[3] では多くのアンチウイルスアプリケーションが配布されている。

本研究では制御フロー解析により Android のマルウェアからパターンを作成して、アプリケーションがどの程度一致するかを数値化することでマルウェアを検出する方法を提案する。単純なバイナリの比較よりもコードの改変に対応でき、既知のマルウェアのパターンで亜種も検出できることを目標とする。

2 関連研究

マルウェアの検出は実際にアプリケーションを実行させて振る舞いから検出を試みる動的解析と、アプリケーションの実行を伴わない静的解析がある。

動的解析では Asaf Shabtai らは Android でマルウェア検出を行うフレームワークの Andromaly[4] を提案している。Andromaly は通常のユーザの振る舞いとは異なる動作が起きたときに機器をロックすることでマルウェアの実行を防ぐ。

静的解析では川端秀明らは Android のアプリケーションを逆コンパイルすることで外部サーバにある JavaScript によって実行される可能性のある脅威を推定する方法 [5] を提案している。

一方、Microsoft Windows ではマルウェアを検出するための方法が多く提案されている。特に静的解析において制御フロー解析を用いた方法では、Halvar Flake がマルウェアのコールグラフや制御フロー解析の結果のグラフを比較することでマルウェアの分類を行う提案 [6] を行っている。また著者らは制御フロー解析の結果のグラフと API 呼び出しに注目したマルウェアの分類を行う提案 [7] をした。

本研究で提案は Microsoft Windows における制御フロー解析を用いたマルウェアの分類方法を Android におけるマルウェア検出に応用したものである。

3 本研究の提案

Android のアプリケーションの多くは Java で作成されており APK 形式で配布されている。APK 形式の実体は Zip 形式のアーカイブである。APK には Dalvik VM の実行形式である classes.dex が含まれている。単純にバイナリを比較することで classes.dex がマルウェアであるか判別する方法も考えられる。しかし同じソースコードであっても異なるバイナリになることもありうる。そのためバイナリレベルの比較ではマルウェアを検出できない可能性がある。

本研究では静的に制御フロー解析を行うことでマルウェアの特徴を抽出し、アプリケーションが抽出した特徴を含んでいるか調べることで、マルウェアを検出する方法を提案する。同じソースコードであれば制御フロー解析の結果は同じになることが期待できる。このときマルウェア単独でアプリケーションを構成している場合には制御フロー解析した結果のすべてを特徴として抽出し、アプリケーションにマルウェアのコードが付け加えられている場合には解析者がマルウェアのコードと判断した部分を特徴として抽出する。

3.1 実装

マルウェア検体の classes.dex を baksmali[8] で逆アセンブルする。baksmali はクラス毎に smali 形式の逆アセンブルファイルを作る。この逆アセンブルファイルをメソッド毎に制御フロー解析する。制御フロー解析を行った結果は、例えば図 1 のグラフになる。呼び出すメソッドがグラフのノードのラベルとなる。このときクラス android/util/Log に属するメソッドの呼び出しはアプリケーションの動作に関係ないので削除する。次に呼び出すメソッドがないノードを削除する。例えば図 1 のラベルを持たないノードを削除すると図 2 のグラフになる。マルウェア検体から作られたグラフのうちマルウェアの動作に必要とされるメソッドをそのマルウェアのパターンとして登録する。パターンはマルウェアの検体毎に作成する。

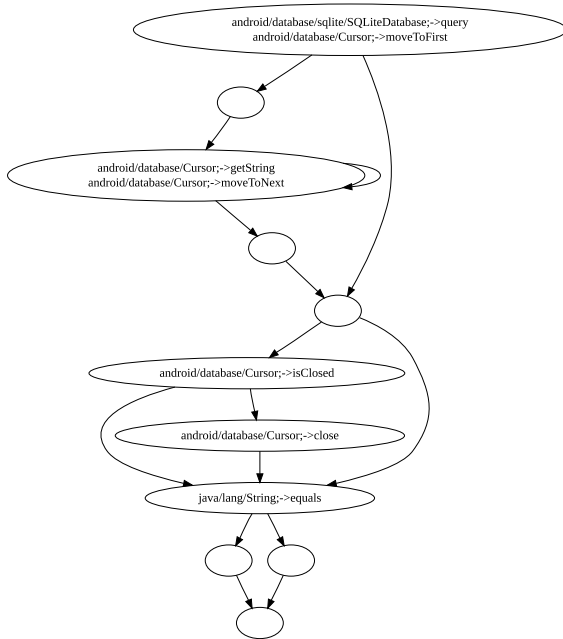


図 1: 制御フロー解析

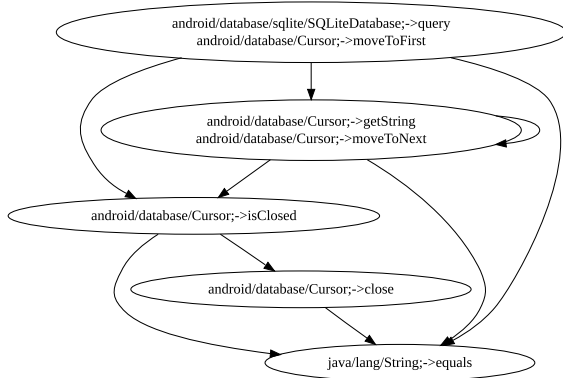


図 2: ノードを削除したグラフ

次に検査するアプリケーションを同様に逆アセンブルして制御フロー解析する。このとき得られたアプリケーションのメソッドの制御フロー解析の結果のグラフとパターンに登録したメソッドのグラフを比較して一致する割合を求め、それをそのマルウェアとアプリケーションの類似度とする。類似度は

$$s = \frac{\text{一致したメソッドのノード数の合計}}{\text{パターンのメソッドのノード数の合計}} \times 100$$

と定義する。類似度が一定の値以上のときにはマルウェアであると判定する。ただし 13 種類のマルウェア検体を解析した結果から考慮すると小さなメソッドが偶然に一致する可能性があ

るので、メソッドのグラフがノードが 1 つしかないか、またはノードの数と呼び出すメソッドの数の合計が 8 未満のいずれかに該当するときには、それらのメソッドをパターンから除く。

4 実験

表 1 の実験環境、解析済みの 13 種類のマルウェア検体、未解析の 14451 種類のアプリケーションを準備した。特定の未解析の 14451 種類のアプリケーションは SHA1 の先頭 6 桁で表す。パターンの名称は Kaspersky または G Data の検出名に基づくが、どのアンチウイルス製品でも検出できなかったマルウェア検体には著者が名前を付けた。

4.1 実験 1

解析済みの 13 種類のマルウェア検体からパターンを作成したところ、2 種類の検体からは同じパターンが抽出された。そのため 12 種類のパターンを作成した。これらのパターンと 14451 種類のアプリケーションの類似度を求めたところ類似度 s の分布は表 2 となった。表 2 では $s = 0$ の数は省かれている。実行時間は制御フロー解析に 1 アプリケーションあたり約 4.09 秒、グラフの比較に約 1.10 秒かかった。また 14451 種類中 38 種類は解析に失敗した。

Adrd.a の類似度が 10 を越えるアプリケーションを解析したところ、Adrd.a の類似度が 80 を越える 2 種類はマルウェアであった。Geinimi の各亜種では類似度が 0 を越えるアプリケーションの合計は 26 であるが、重複するものを除くとすべての亜種で共通して類似度が 0 ではない 6 種類と Geinimi.e だけで類似度が 1 となった 2 種類だけである。Geinimi の各亜種で類似度が 0 を越える 8 種類のアプリケーションを解析し

CPU	Pentium M 1.20GHz
Memory	1GB
OS	Ubuntu 10.04 LTS

表 1: 実験環境

	$0 < s \leq 10$	$10 < s \leq 20$	$20 < s \leq 30$	$30 < s \leq 40$	$80 < s \leq 90$
Adrd.a	28	7			2
FakePlayer.a					
FakePlayer.b					
FakePlayer.c					
FakePlayer.h					
Flexispy.a	112				
Geinimi.a	2	4			
Geinimi.b	2	4			
Geinimi.c	2	4			
Geinimi.e	4	4			
Tapsnake.a	3			3	
TheftAware.a	371		1		

表 2: 実験 1 の結果

たところ, Geinimi のすべての亜種に共通する 6 種類はマルウェアであった. また TheftAware.a の類似度が 24 となったアプリケーションを解析したところ, マルウェアであった.

一方, Flexispy.a の一部と Tapsnake.a の類似度が高いアプリケーションを解析したがマルウェアを見つけることはできなかった. Tapsnake.a のコードはマルウェアとは関係ないアプリケーションに含まれているが, パターンを作成する時に誤ってアプリケーションの部分もパターンに含めてしまった. そのためマルウェアを含まないアプリケーションの類似度が 39 という高い値になった.

この結果から 8784ee, cbf6ba の 2 種類が Adrd.a の亜種であり, 135919, 632153, 6889d7, 8b9d22, bc894f, f3ed5f の 6 種類が Geinimi の亜種, a20a20 が TheftAware.a の亜種であると判断した.

4.2 アンチウイルス製品による検出

表 3 は 14451 種類のアプリケーションを G Data, Kaspersky, Symantec, Trend Micro の Windows 版アンチウイルス製品で検査を行った結果である. ただし本研究では classes.dex を対象としているので, classes.dex 以外のデータをマルウェアとして検出している場合は除いた. なお表 3 は 2011 年 7 月 21 日時点での結果である.

4.3 実験 2

実験 1 で新たに見つかったマルウェアおよびアンチウイルス製品によって検出されたマルウェアから特徴を抽出しパターンを作成した. 全部で 26 種類あるがパターンを作成したところ次の組み合わせでは同一のパターンになったためパターンの数は 15 種類である.

- 059ed6, cf2270, f84f82
- 1fad7a, 64bba3
- 256347, d29469
- 61be35, c3b6c9
- 6889d7, 8b9d22, bc894f, f3ed5f
- 91deec, ac9532, aed5e0, dca9e4

また実験 1 で誤って作成した Tapsnake.a のパターンを修正した. これを含めた 16 種類のパターンで実験 1 の制御フロー解析の結果から同様に類似度を求めたところ類似度 s の分布は表 4 となった. 表 4 では $s = 0$ の数は省かれている. 実行時間はグラフの比較に 1 アプリケーションあたり約 1.10 秒かかった.

KungFu.z で類似度が 65 になったアプリケーションを除いて類似度が 20 を越えるアプリケーションはすべて実験 1 またはアンチウイルス製品で見つかったマルウェア検体であった. Adrd.c, Adrd.s, Adrd.ab, Adrd.ak, Adrd.ap は類似度が 70 を越えるアプリケーションがそれぞれ 7 種類あったが, この 7 種類はアンチウイルス製品が Adrd の亜種として検出したマルウェア検体であった. Adrd.8784ee, Adrd.cbf6ba の類似度が 90 を越えるアプリケーションは 8784ee と cbf6ba であった. Geinimi.y, Geinimi.z, Geinimi.ar も類似度が 20 を越えるアプリケーションがそれぞれ 6 種類あったが, この 6 種類は実験 1 で Geinimi の亜種と判断したマルウェア検体であった.

一方, 類似度が 10 を越えるアプリケーションのうちいくつかを解析したがマルウェアではなかった.

KungFu.z は通常のアプリケーションを改変して作成されており, 類似度が 65 になったア

	G Data	Kaspersky	Symantec	Trend Micro
059ed6		Trojan-SMS.AndroidOS.Fakelogo.s		
135919	Android:Geimini-C [Trj] (Engine-B)	Trojan-Spy.AndroidOS.Geimini.ar		
1700f4	Android:Bosm-A [Trj] (Engine-B)			AndroidOS.BOMB.S
1fad7a	Android:Adrd-B [Trj] (Engine-B)	Trojan-Spy.AndroidOS.Adrd.ap		AndroidOS.PJAPPS.D
256347	Android:Adrd-C [Trj] (Engine-B)			
465559	Android:Trojan.DroidKungFu4.C (Engine-A)	Backdoor.AndroidOS.KungFu.z		
5c8ac0	Android:Adrd-C [Trj] (Engine-B)	Trojan-Spy.AndroidOS.Adrd.ak	Android.Pjapps	AndroidOS.ADRD.D
61be35	Android:Trojan.PjApps2.E (Engine-A)	Trojan-Spy.AndroidOS.Adrd.c	Android.Pjapps	AndroidOS.PJAPPS.B
632153	Android:Geimini-C [Trj] (Engine-B)	Trojan-Spy.AndroidOS.Geimini.z	Android.Geimini	AndroidOS.GEINIMI.D
64bba3	Android:Adrd-B [Trj] (Engine-B)	Trojan-Spy.AndroidOS.Adrd.ap		AndroidOS.PJAPPS.D
6889d7	Android:Geimini-C [Trj] (Engine-B)	Trojan-Spy.AndroidOS.Geimini.y	Android.Geimini	AndroidOS.GEINIMI.N
8b9d22	Android:Geimini-B [Trj] (Engine-B)			
91deec	Android:Trojan.PjApps2.A (Engine-A)		Android.Geimini	AndroidOS.PJAPPS.H
ac9532	Android:Trojan.PjApps2.A (Engine-A)	Trojan-Spy.AndroidOS.Adrd.s	Android.Adrd	AndroidOS.ADRD.AL
aed5e0	Android:Adrd-C [Trj] (Engine-B)			ANDROIDOS.PJAPPS.H
bc894f	Android:Geimini-B [Trj] (Engine-B)			AndroidOS.GEINIMI.B
c3b6c9	Android:Adrd-B [Trj] (Engine-B)			
c3d4a2	Android:Trojan.DroidKungFu.B (Engine-A)	Backdoor.AndroidOS.KungFu.a	Android.Fokonge	AndroidOS.GONFU.A
cf2270		Trojan-SMS.AndroidOS.Fakelogo.s		
d29469	Android:Adrd-C [Trj] (Engine-B)	Trojan-Spy.AndroidOS.Adrd.ab	Android.Pjapps	
dca9e4	Android:Adrd-C [Trj] (Engine-B)	Trojan-Spy.AndroidOS.Adrd.aa	Android.Pjapps	AndroidOS.PJAP.A
f3ed5f	Android:Geimini-B [Trj] (Engine-B)			
f84f82		Trojan-SMS.AndroidOS.Fakelogo.s		

表 3: アンチウイルス製品の検出結果

アプリケーションはKungFu.zの元になった通常
のアプリケーションであった。KungFu.zのパ
ターンを作成するとき、改変されていないメソ
ッドもパターンに含めてしまったため、元のアプ
リケーションで類似度が大きくなった。

5 考察

本研究で提案する手法で類似度から技術者が
解析すべき検体を特定することはでき、マルウエ
アを見つけることはできた。特に Adrd と Gein
imi では特徴を抽出した検体以外の亜種も類似
度が高くなった。このことから既存のマルウエ
アから抽出した特徴を用いて、亜種も見つける
ことが可能であり、マルウェア検体入手・解
析していない亜種にも対応できると言える。

一方、共通するコードがない場合にはマルウエ
アを見つけることはできなかった。そのため新
規に作成されたマルウェアには提案する手法を
用いることはできない。

マルウェアの特徴として抽出したメソッドの
うちのいくつかがマルウェア以外でも使われる
一般的なメソッドであったため、マルウェアで
なくても類似度が0より大きいアプリケーション
があった。また類似度にはパターン毎にばら
つきがあった。Geimini ではマルウェアと判断
したアプリケーションの中で最小の類似度は7
であり、マルウェアではないアプリケーション
で最も高かった類似度は4であった。しかし他
のパターンでは7以上であってもマルウェアで

はないアプリケーションがあった。これらの理
由から類似度の大小だけを見てマルウェアを判
定することはできなかった。

本研究ではマルウェアを技術者が解析し、そ
の結果に基づいて特徴を抽出してパターンの作
成したが、その作業には失敗があった。実験1で
はTapsnake.aのパターンにマルウェア以外のメ
ソッドを含めてしまった。実験2ではKungFu.z
のパターンに元になった通常のアPLICKEI
ションの改変されていないメソッドも含めてしま
った。特に実験2の事例ではマルウェアとそうで
ないメソッドが混在しているため、技術者がマ
ルウェアではないメソッドをパターンから取り
除くことは困難である。

また解析に失敗した38種類はファイルの破
損もしくは逆アセンブラが対応していないこと
が原因であると思われる。

6 今後の課題

本研究の結果では一般的なスマートフォン
ユーザが類似度を見てアプリケーションがマル
ウェアであるか否か判断することはできない。
マルウェアであるか否かを決める方法として、
パターン毎にパターンを作成した技術者が、類
似度がある値を上回ったらマルウェアであると
判定する閾値を設定する方法が考えられる。例
えばGeimini.aの場合にはこの閾値は7になる。
マルウェアの解析を行う技術者に対して指針を
与えるという段階から、一般的なスマートフォ

	0 < s ≤ 10	10 < s ≤ 20	20 < s ≤ 30	30 < s ≤ 40	40 < s ≤ 50	50 < s ≤ 60	60 < s ≤ 70	70 < s ≤ 80	80 < s ≤ 90	90 < s ≤ 100
Adrd.c	4							3	2	2
Adrd.s	7									4
Adrd.ab	4							4	1	2
Adrd.ak	4							6		1
Adrd.ap	4							3	2	2
Adrd.8784ee	29	7								2
Adrd.cbf6ba	29	7								2
Bosm-A										1
Fakelogo.s										3
Geinimi.y	6		2							4
Geinimi.z				4						2
Geinimi.ar				4						2
KungFu.a	143									1
KungFu.z	7					1				1
Tapsnake.a										
TheftAware.a20a20	470									1

表 4: 実験 2 の結果

ンユーザがアプリケーションがマルウェアであるか否か判定を求めるという段階に発展させるためには類似度と閾値について検討する必要がある。

また 2 回の実験からパターンの作成が困難であることがわかった。解決方法としてはマルウェアではないアプリケーションのメソッドのグラフを集めたホワイトリストを作ることが考えられる。技術者が解析の結果からマルウェアの動作に必要なメソッドを抽出し、その中からホワイトリストに一致するメソッドを取り除き、残りをマルウェアの特徴とする方法が検討できると思われる。

本研究では偶然にメソッドのグラフが一致することを防ぐために、小さなメソッドを除外する条件を設けた。ホワイトリストを用いるならば、この小さなメソッドを除外する条件は不要かもしれない。小さなメソッドを除外する場合でも条件の再検討が必要である。

本研究の実験は PC を用いたが、スマートフォンユーザが利用するためにはスマートフォンで動作するアプリケーションとして実装する必要がある。

謝辞 本研究の一部は科学研究費 (23500174) の助成を受けたものである。本研究を行うにあたって日本コンピュータセキュリティリサーチ株式会社主席研究員の遠藤基氏の協力を得た。

参考文献

- [1] 情報処理推進機構, IPA テクニカルウォッチ スマートフォンへの脅威と対策に関するレポート, <http://www.ipa.go.jp/about/technicalwatch/20110622.html>
- [2] McAfee Labs, McAfee 脅威レポート 2011 年第 1 四半期, <http://www.mcafee.com/japan/security/threatreport11q1.asp>
- [3] <https://market.android.com/>
- [4] <http://andromaly.wordpress.com/>
- [5] 川端秀明, 磯原隆将, 竹森敬祐, 窪田歩, Android パーミッションを悪用する Script の脅威と静的解析, 情報処理学会研究報告 Vol.2011-CSEC-53 No.3
- [6] Halvar Flake, Automated Unpacking and Malware Classification, Black Hat Japan 2007 pp.61-88, 25th October 2007
- [7] 岩本一樹, 和崎克己, 静的解析によるマルウェアの分類と結果の検討, マルチメディア, 分散, 協調とモバイル (DICOMO2010) シンポジウム 頁:477-491
- [8] <http://code.google.com/p/smali/>