Vol.2013-CSEC-61 No.5
Vol.2013-IOT-21 No.5
2013/5/9

制御フロー解析により生成されたグラフ比較による Androidマルウェア検出方法の提案

岩本 一樹[1,a)]　和﨑 克己[1,b)]

概要：Android のマルウェアは大幅に増加しているため，マルウェアを効率よく検出できる方法が必要である．本研究では Android のアプリを制御フロー解析してメソッド毎のグラフを生成し，そのグラフを比較することでマルウェアを検出する方法を提案する．提案する方法ではマルウェアから生成したグラフを Android のアプリが含んでいる割合を包含度で表し，包含度が閾値を超えた場合にはマルウェアとして検出する．我々は制御フロー解析の結果をそのまま用いるアルゴリズムと，最適化や難読化を考慮したアルゴリズムを実装した．これら 2 つのアルゴリズムでマルウェアから生成したグラフと収集した Android のアプリから生成したグラフを比較することで包含度を求め，それぞれの 2 つのアルゴリズムの性能を検討した．結果，我々が提案する方法で収集した Android のアプリの中からマルウェアを検出することができた．

Detection Method for Android Malware by comparing Control Flow Graphs

Kazuki Iwamoto[1,a)]　Katsumi Wasaki[1,b)]

Abstract: The effective method to detect Android malware is necessary, because the number of Android malware has been increasing. In this paper, we propose the method to detect malware by comparing the graphs generated from Android applications by control flow analysis. We defined inclusion degree as the ratio including the graph extracted from malware. And an application is detected if inclusion degree is greater than threshold. We implemented 2 algorithms. The first algorithm uses the results of control flow analysis directly. The second algorithm corresponds to the optimization and obfuscation. We got inclusion degrees by comparing the graphs generated from Android applications with the graphs generated from malware, and considered the performances of those algorithms. Finally, we detected some malware in Android applications.

1. はじめに

Android を対象とするマルウェアは 2012 年に入って大幅に増加している [1]．それらすべてを技術者が解析することは困難であるため，マルウェアを効率よく検出できる方法が必要である．ゆえに本研究は既存のマルウェアを改変した亜種の検出や難読化されたマルウェアを，既存のマルウェアから抽出した特徴と比較することで検出することを試みる．

本研究では制御フロー解析により，Android のアプリからメソッドごとのグラフを生成し，そのグラフを比較することでマルウェアの検出を試みる方法を提案する．わずかな改変や難読化によってコードに違いがあっても，コードの意味が同じならばその違いは生成されるグラフに反映されず，正しく比較が行えることが期待できる．グラフを比較するにあたり，マルウェアから生成したグラフを Android のアプリがどれだけ含んでいるかを包含度（Inclusion degree）で表す．あるアプリがマルウェアから生成したグラフを完全に含んでいるならば包含度は 100%となり，そのアプリはマルウェアといえる．また包含度がある一定の値（閾値）を超えたときには，そのアプリを技術者が解析すべきアプリとする．この包含度と閾値を用いることで，一部の

[1] 信州大学大学院総合工学系研究科
Interdisciplinary Graduate School of Science and Technology, Shinshu University, Matsumoto, Nagano, 390-8621
[a)] iwm@maid.org
[b)] wasaki@cs.shinshu-u.ac.jp

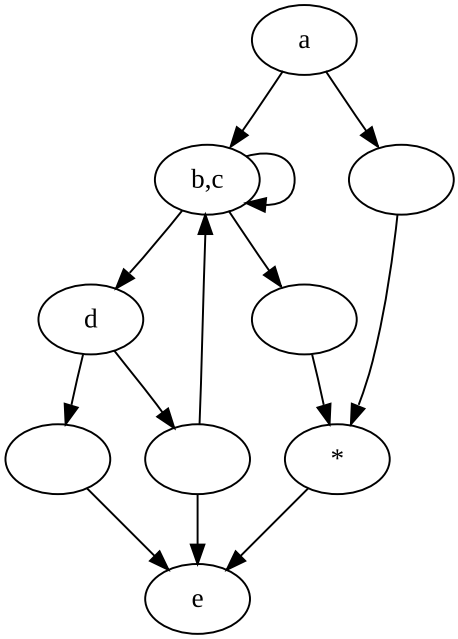

図 1　制御フロー解析
Fig. 1　Control flow analysis

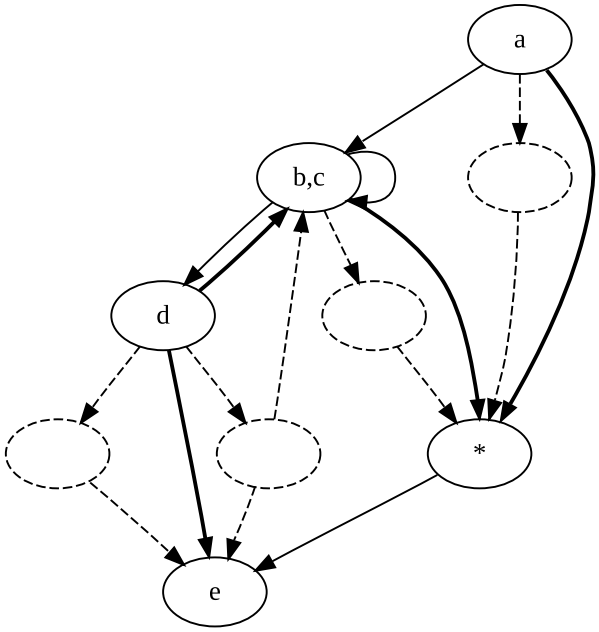

図 2　削除されるノードと追加されるエッジ
Fig. 2　Removed nodes and appended edges

メソッドだけが改変されたマルウェアに対応する．

2. 本研究の提案

本研究では 2 つのアルゴリズムを提案する．

2.1 第 0 アルゴリズム

Android のアプリのファイルの拡張子は.apk であるが，これは ZIP 形式のアーカイブであり，このアーカイブの中に Dalvik VM の実行形式である classes.dex が含まれている．classes.dex は baksmali[*1] で逆アセンブルすることができる．本研究では逆アセンブル結果に対して，メソッドごとに制御フロー解析を行いグラフを生成する．このグラフのノードのラベルは呼び出すメソッドの名前になる．ただしアプリ内部のメソッド呼び出しの場合，その名前はアプリ作成時に変えることができるので，メソッドの名前を「*」とする．またクラス android/util/Log に属するメソッドはアプリの動作には関係ないので削除する．このとき呼び出すメソッドがないノードは削除される．

たとえば制御フロー解析を行った結果は**図 1** のグラフになる．図 1 では**図 2** で点線で示した呼び出すメソッドがないノードとエッジは削除され，太線で示したエッジが作成される．結果的に図 1 は**図 3** のラベル付きグラフになる．

本研究ではマルウェア検体から生成されたグラフのうち，マルウェアの動作に必要とされるメソッドのグラフをそのマルウェアの定義ファイルとする．

検査するアプリに対しても同様に逆アセンブルして制御フロー解析を行いグラフを生成する．このとき得られたアプリのグラフに定義ファイルのグラフが含まれる割合を求め，その値をそのマルウェアの包含度とする．包含度は

$$I = \frac{\text{一致したグラフのノード数の合計}}{\text{定義ファイルのグラフのノード数の合計}} \times 100$$

と定義する．包含度が 0 ならば検査対象のアプリは定義ファイルのグラフを全く含んでおらず，包含度が 100 ならば定義ファイルのグラフをすべて含んでいる．

第 0 アルゴリズムは文献 [2] で提案した方法を基にしている．ただし文献 [2] ではメソッドのグラフのノードが 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いていたが，本研究ではそのような除外は行わない．

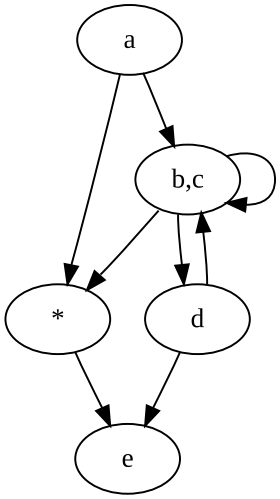

図 3　ノードを削除したグラフ
Fig. 3　Shrinked graph

[*1] http://code.google.com/p/smali/

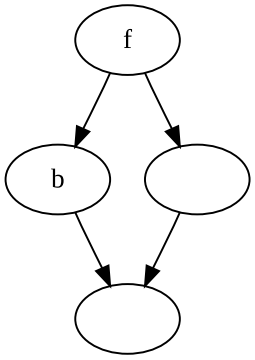

図 4　呼び出されるメソッド
Fig. 4　Invoked method

2.2　第 1 アルゴリズム

第 1 アルゴリズムは第 0 アルゴリズムを改変したものであり，最適化や難読化を想定している．最適化や難読化では次のことが考えられる．

(1) 識別子（クラス名，メソッド名，変数名など）を無意味な文字列に置き換える
(2) 文字列などのデータを暗号化して実行時に復号する
(3) 制御フローの変化
　(a) 実行されないコードを削除する
　(b) 無意味なコードを追加する
　(c) メソッド呼び出しをインラインに展開する
　(d) コードを分断してメソッドにする

名前を変更することのできない外部のメソッド呼び出しを除いて，第 0 アルゴリズムでは識別子を扱っていない．またデータも扱っていないので（1）と（2）は本研究には関係ない．ゆえに（3）のみを検討する．

（3a）と（3b）では，メソッド全体が削除または追加されるならば，包含度が減少することになる．単純にコードが削除されたり追加されるだけならば，第 0 アルゴリズムのラベルを持たないノードを削除することで対応できる可能性がある．しかしコードのメソッドの呼び出し自体が削除または追加されたときの対応は難しい．

第 1 アルゴリズムでは（3c）と（3d）についてのみ考慮する．（3c）と（3d）は反対の操作であり，コードが分断されたならばインライン展開することで元に戻せる可能性がある．一方，インライン展開されたコードを特定して再び独立したメソッドに戻すことは難しい．そこで，コードを分断した場合には分断元から呼び出されるだけなので，1 箇所から呼び出されているメソッドをすべて探してインライン展開することで，可能な限り大きなグラフに収束させる．

たとえば図 1 の「*」で呼び出されるメソッドが**図 4** のとき，図 4 を図 1 に展開すると**図 5** になる．図 5 では展開されたノードとエッジは太線で示される．

第 0 アルゴリズムと同様に第 1 アルゴリズムでもメソッドの呼び出しがないノードを削除し，マルウェアのメソッドから生成されたグラフを定義ファイルとする．同様に検査するアプリもインライン展開し，定義ファイルと比較して包含度を求める．包含度の定義も第 0 アルゴリズムと同様である．

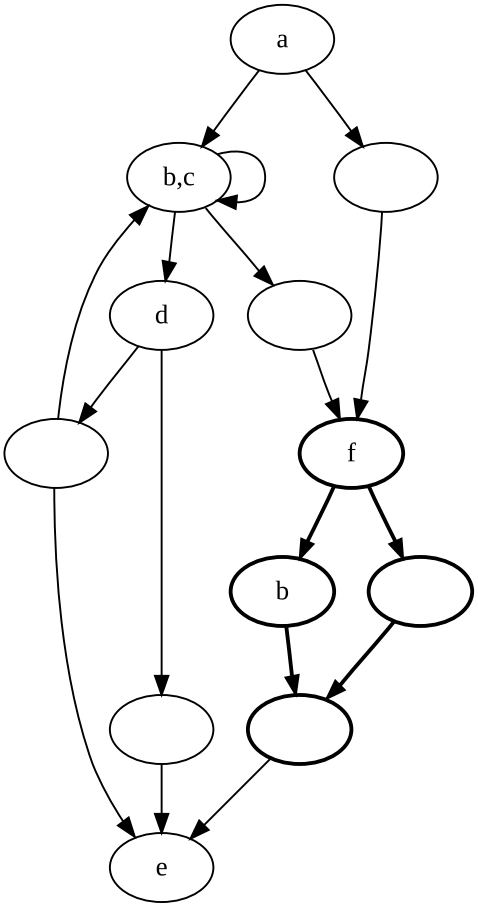

図 5　インライン展開
Fig. 5　Inlining

2.3　閾値

定義ファイルのグラフと検査対象のアプリから生成したグラフが偶然に一致する場合もありうるので，包含度が 0 よりも大きい場合でもアプリがマルウェアであるとは限らない．しかし一致するグラフがある程度あるならばマルウェアである可能性があるので，包含度が一定の値（閾値）を超えたならば技術者が解析すべきアプリとする．

本研究では明らかにマルウェアではないアプリと定義ファイルを比較したときの包含度の最大値から閾値を求める．

3.　実験

本研究で扱うアプリは次の 5 つの状態のうち 1 つに属する．

- `Black`　どちらかのアルゴリズムで包含度が

100 である
- `Gray` どちらかのアルゴリズムで包含度が閾値より大きいが包含度は 100 ではない
- `Ivory` どのアルゴリズムでも包含度が閾値以下である
- `White` マルウェアではないことが明らかで閾値を算出する対象となるアプリ
- `Failure` 解析不可能なアプリ

本研究では毎月，集めた検体に対して次の作業を行う．

(1) 90 日以上前の Ivory のアプリを White に移動して閾値を更新する．
(2) 当月分のアプリと Black または Gray，Ivory の未解析のアプリを検査する．検査結果によりアプリの属性が変更される．
(3) Gray のアプリを解析し，Black または Ivory に移動する．Ivory のアプリの中にマルウェアがあるならば，それを Black に移動する．マルウェアであるときには定義ファイルを改変・追加する．

定義ファイルは 1 つ以上のマルウェアのメソッドを制御フロー解析した結果のグラフから作られる．また定義ファイルはマルウェアに科名（Family name）と亜種名（Variant name）で構成される名前を与える．たとえば名前が Adrd.a ならば科名は Adrd，亜種名は a である．

表 1 は月ごとに本研究で提案する 2 つのアルゴリズムでマルウェアの検出を試みたときの各状態のアプリの数である．初期の 13 種類のアプリと 2011 年 6 月の 14,451 種類のアプリは文献 [2] で用いたアプリである．表 1 の Definition では下段に既存の定義ファイルを改変した数，新規に加えた数，削除した数を示す．Black と Gray，Ivory の括弧内の数値は誤ってその状態になった数であり，Black と Gray ではマルウェアではないアプリの数，Ivory ではマルウェアであるアプリの数である．Black と Gray では，どちらか一方または両方のアルゴリズムで検出した数を下段に示す．Ivory では検出できなかった検体の数と科名の数，既存の定義ファイルと同一の科名をもつ検体の数と科名の数を下段の括弧内に示す．

表 2 は誤って Gray となったアプリを検出した定義ファイルとアプリの数である．**表 3** は検出できずに Ivory となったマルウェアの中で，既に同じ科名のマルウェアがある検体の科名と検体数である．

3.1 閾値

2011 年 6 月では White のアプリがないため閾値を算出することができないので，包含度が 0 ではないすべてのアプリを Gray にした．定義ファイルには偶然に一致する可能性がある小さなグラフも含まれているため，ほとんどのアプリで包含度が 0 を超えた．

表 2 偽陽性
Table 2 False positive

Name	Sample
DorDrae.a	2
FakeInst.ed	1
Kidlogger.e	5
Kiser.a	1
Kiser.c	11
Kiser.f	11
Twofor.b	1

表 3 偽陰性
Table 3 False negative

Family Name	Sample
Agent	52
Anti	1
Bosm	1
Boxer	4
Copycat	10
Cosha	1
DorDrae	4
FaceNiff	1
FakeDoc	1
FakeInst	46
GinMaster	42
Hamob	91
Jifake	6
Kiser	1
Kmin	246
KungFu	7
LdPinch	1
Nyleaker	1
Opfake	22
Penetho	1
Plangton	4
SMSreg	10
SmsSpy	1
Twofor	1
Whapsni	2

2011 年 9 月では White のアプリの中で定義ファイルごとに最大の包含度を求めてそれを閾値とした．しかし Gray の中でマルウェアは 10 種類であるのに対して誤検出が 77 種類あった．2011 年 9 月の時点で，検出した亜種が存在するマルウェアをその亜種の定義ファイルで比較したときに，包含度が最大となる亜種の名前とその包含度，White アプリ中の最大の包含度，包含度の比率を**表 4** と**表 5** に示す．

表 4 と表 5 から，2011 年 11 月以降は亜種を定義ファイルで検出できた場合には，亜種を検出したときの包含度の最小値と White アプリ中の最大の包含度の中間の値を閾値とする．亜種を検出できない場合や亜種がない場合には，亜種を検出したときの包含度の代わりにこの値を 100 とする．ただし表 4 の FakePlayer.b および FakePlayer.h の例を考慮して，White アプリ中の最大の包含度の 1.5 倍を閾値の最低値とする．

亜種を検出したときの包含度を I_{vi}，White のアプリの中で最大の包含度を I_w とすると閾値 T は

$$I = \begin{cases} \min\{I_{v1}, I_{v2}, \ldots, I_{vn}\} & (\text{亜種を検出時}) \\ 100 & (\text{上記以外}) \end{cases}$$

$$T = \max(\frac{I + I_w}{2}, I_w \times 1.5)$$

表 1　マルウェア検出
Table 1　Malware detection

Date	Definition			Black			Gray			Ivory				White	Failure
	Fix	Add	Remove	0th	1st	Both	0th	1st	Both	Sample	Family	Sample(Variant)	Family(Variant)		
Initial		11		0	13 0	13		0			0			0	0
2011/6		11		0	13 0	13	17(17)	14,400(14,353) 0	14,383 (14,336)		16			0	35
2011/9	4	34 23	0	0	74 2	72	67(66)	87(77) 9(8)	11(3)	(13	14,165 8	1	1)	14,369	75
2011/11	2	46 12	0	0	99 2	97	1(1)	9(1) 0	8	(263	26,304 12	247	2)	14,369	82
2011/12	9	65 19	0	6	372 2	364	0	2(2) 0	2(2)	(6	14,762 6	2	2)	28,598	84
2012/2	0	66 6	5	7	385 2	376	1(1)	12(1) 0	11	(75	12,623 8	48	3)	40,411	242
2012/3	2	79 14	1	20	475 5	450	0	2(1) 1(1)	1	(52	14,556 3	3	1)	43,356	395
2012/4	0	83 5	1	99	613 16	498	10(10)	10(10) 0	0	(19	22,601 12	15	7)	43,367	489
2012/5	2	91 9	1	125	671 18	524	1	1 0	0	(43	18,547 4	2	1)	52,944	571
2012/6	0	97 6	0	175	739 24	536	0	1 0	1	(126	18,027 10	111	7)	57,862	684
2012/7	0	112 15	0	245	907 24	634	10(7)	14(7) 0	4	(2	15,171 2	2	2)	65,922	720
2012/8	3	114 2	0	271	1,002 31	695	4(3)	12(4) 0	8(1)	(62	17,175 8	59	5)	71,409	880
2012/9	3	127 13	0	277	1,166 82	798	0	1 0	1	(1	17,806 1	1	1)	75,767	943
2012/10	1	127 0	0	285	1,274 133	846	0	1 0	1	(9	15,295 7	3	3)	81,102	998
2012/11	0	136 9	0	300	1,404 165	929	5(1)	17(5) 12(4)	0	(48	15,511 8	36	3)	88,526	1,106
2012/12	0	147 11	0	358	1,542 190	981		0		(15	15,649 4	15	4)	93,571	1,157
2013/1	0	153 6	0	368	1,753 260	1112	1(1)	2(1) 0	1	(5	15,774 2	4	1)	96,384	1,199
2013/2	1	155 2	0	371	1,834 298	1152	6	8 0	2	(10	10,956 6	8	5)	103,990	1,240
2013/3	1	163 8	0	390	1,975 344	1228	0	3 0	3		9,953			109,200	1,292
Latest	2	163 0	0	390	1,978 344	1231		0			9,953			109,200	1,292

で求める．また White の中からランダムに 500 種類のアプリを選び再検査することで，新しく追加した定義ファイルの閾値を決める．

たとえば表 4 の Adrd.ca では亜種を検出したときの包含度は Adrd.a で 72，Adrd.bw で 93 なので

表 4　亜種の包含度（第 0 アルゴリズム）
Table 4　Variant inclusion (0th algorithm)

Detected name	Variant name	Inclusion degree (Variant)	Inclusion degree (White)	Rate
Adrd.a	Adrd.ca	72	18	4
Adrd.c	Adrd.ap	84	4	21
Adrd.s	Adrd.ab	33	4	8.25
Adrd.ab	Adrd.ak	83	4	20.75
Adrd.ak	Adrd.ab	85	4	21.25
Adrd.ap	Adrd.c	83	4	20.75
Adrd.bw	Adrd.ca	93	18	5.17
Adrd.ca	Adrd.bw	95	18	5.28
FaceNiff.a	FaceNiff.b	9	8	1.12
FaceNiff.b	FaceNiff.c	96	7	13.71
FaceNiff.c	FaceNiff.b	95	8	11.88
FakePlayer.a	FakePlayer.b	66	0	—
FakePlayer.b	FakePlayer.a	66	33	2
FakePlayer.h	FakePlayer.a	66	33	2
Geinimi.a	Geinimi.b	93	6	15.5
Geinimi.a	Geinimi.y	90	7	12.86
Geinimi.b	Geinimi.a	95	13	7.31
Geinimi.c	Geinimi.a	98	13	7.54
Geinimi.e	Geinimi.a	95	13	7.31
Geinimi.y	Geinimi.a	96	13	7.38
Geinimi.z	Geinimi.z	94	8	11.75
Geinimi.ar	Geinimi.ar	94	8	11.75
Geinimi.bw	Geinimi.a	97	13	7.46
KungFu.a	KungFu.aj	3	5	0.6
KungFu.aj	KungFu.cl	37	8	4.62
KungFu.cl	KungFu.aj	34	5	6.8
SeaWeth.a	SeaWeth.b	36	5	7.2
SeaWeth.b	SeaWeth.a	84	6	14

表 5　亜種の包含度（第 1 アルゴリズム）
Table 5　Variant inclusion (1th algorithm)

Detected name	Variant name	Inclusion degree (Variant)	Inclusion degree (White)	Rate
Adrd.a	Adrd.ca	59	13	4.54
Adrd.c	Adrd.ap	84	3	28
Adrd.s	Adrd.ab	29	3	9.67
Adrd.ab	Adrd.ak	83	3	27.67
Adrd.ak	Adrd.ab	85	3	28.33
Adrd.ap	Adrd.c	82	3	27.33
Adrd.bw	Adrd.ca	92	13	7.08
Adrd.ca	Adrd.bw	93	13	7.15
FaceNiff.a	FaceNiff.b	4	6	0.67
FaceNiff.b	FaceNiff.c	96	4	24
FaceNiff.c	FaceNiff.b	95	6	15.83
FakePlayer.a	————	–	–	—
FakePlayer.b	FakePlayer.a	23	23	1
FakePlayer.h	FakePlayer.a	23	23	1
Geinimi.a	Geinimi.c	84	5	16.8
Geinimi.a	Geinimi.y	85	6	14.17
Geinimi.b	Geinimi.a	88	14	6.29
Geinimi.c	Geinimi.a	99	14	7.07
Geinimi.e	Geinimi.a	96	14	6.86
Geinimi.y	Geinimi.a	94	14	6.71
Geinimi.z	Geinimi.ar	93	6	15.5
Geinimi.ar	Geinimi.z	93	6	15.5
Geinimi.bw	Geinimi.a	89	14	6.36
KungFu.a	KungFu.aj	2	6	0.33
KungFu.aj	KungFu.cl	31	7	4.43
KungFu.cl	KungFu.aj	28	6	4.67
SeaWeth.a	SeaWeth.b	32	3	10.67
SeaWeth.b	SeaWeth.a	78	4	19.5

$$T = \max(\frac{\min(72, 93) + 18}{2}, 18 \times 1.5) = 45$$

になる．また Adrd.a の定義ファイルでは亜種のなかで包含度が最大になることはなかったので，

$$T = \max(\frac{100 + 19}{2}, 19 \times 1.5) = 59$$

になる．KungFu.aj では $I \leq I_w$ なので閾値は $T = 7$ になる．

3.2　定義ファイル

定義ファイルは解析者によって主に次の 3 つの方法で作成される．

(1) マルウェア固有のメソッドまたはクラスを指定する
(2) 複数の同じ種類のマルウェア検体に共通するメソッドを求める
(3) マルウェア検体の全メソッドから通常のアプリのメソッドを取り除く

これらの方法は完全に独立してはおらず，これらの方法が組み合わされる．もっとも基本的な方法は（1）であり，解析者が指定したメソッドだけが定義ファイルに含まれる．またクラスを指定したときには，そのクラスに属するメソッドが定義ファイルに含まれる．検体が複数ありそれらが 1 つの定義ファイルで同じ名前で検出されるべきと判断したならば，（2）の方法を用いることができる．通常のアプリが改造されたり，マルウェアのコードが追加されているならば，（3）の方法を用いることもできる．

初期の 13 種類の検体に対して文献 [2] では 12 種類のパ

ターンを作成していた．しかし本研究では FakePlayer.c の定義ファイルは FakePlayer.b の定義ファイルのすべてのグラフを含んでいるので，FakePlayer.b の定義ファイルだけを作成した．この場合，FakePlayer.c の包含度が 100 ならば必ず FakePlayer.b の包含度も 100 になるので，検出を目的とするならば FakePlayer.c の定義ファイルは必要ない．2012 年 2 月と 2012 年 3 月，2012 年 5 月も同様にある定義ファイルが別の定義ファイルを含むことが明らかになったので定義ファイルを削除した．

2012 年 4 月には今まで SmsSpy.a として検出してきた検体をマルウェアではないと判断して定義ファイルを削除した．そのため Black に属していた 11 種類のアプリを White に移動させた．

また新たなマルウェア検体が見つかったことで，既存の定義ファイルの修正も行っている．これは新たなマルウェア検体の包含度を 100 にするために，新たなマルウェア検体と既知のマルウェア検体に共通しないグラフを定義ファイルから削除したこと（2），および正常なアプリと重複するグラフを定義ファイルから削除したこと（3）が原因である．

2012 年 5 月以降，Black のアプリの総数と個別のアルゴリズムの検出数が一致していない．これは本研究で提案するアルゴリズムではなく，技術者による解析によりアプリを Black に固定したことが原因である．

4. 考察

Black となった検体のうち 13 種類は本研究で提案する方法で検出したものではない．これらのうち 10 種類は，そのグラフのすべてが通常のアプリと一致するため，本研究で提案する方法で対応できなかった．それ以外は解析にかかる時間から定義ファイルの作成を諦めた検体である．

また本研究の提案ではネイティブコードや脆弱性を攻撃するデータファイルなどは対象外である．これらの検出には，他の方法を検討する必要がある．

4.1 定義ファイル

どのメソッドを定義ファイルに含めるのか，あるいは既存の定義ファイルを改変するのか新しい定義ファイルを追加するのかなどの判断は解析者が行った．解析者の技術力や経験に依存しており表 1 にある通り，初期には定義ファイルの修正や削除が比較的多くあった．

難読化対策のために新たに実装した第 1 アルゴリズムが，その目的とは異なる働きがみられた．第 3.2 項の（2）の方法で定義ファイルを作るとき，コードがインライン展開されたために第 0 アルゴリズムで共通していたグラフが第 1 アルゴリズムでは一致しなくなった例があった．そのため定義ファイルに含まれるグラフの数が減り，結果として包含度が 100 になる可能性が高くなったことで第 1 アルゴリズムだけでの検出が発生することになった．

これは定義ファイル作成の段階で含める必要がないグラフを削除できたと解釈することもできる．意図した結果ではないがインライン展開が有効に機能したといえる．

4.2 検出性能の評価

Black となった検体のうち 1,231 種類は両アルゴリズムにより検出できたが，どちらかのアルゴリズムだけで検出したマルウェアも少なくない．仮に我々の過去の提案 [2] のように単一のアルゴリズムしか実装していなかったならば検出率は低下することは明らかであり，2 つのアルゴリズムを実装したのは成功であった．

閾値の算出方法が定まった 2011 年 11 月以降，両アルゴリズムで Gray となったアプリは 43 種類あり，そのうち 3 種類はマルウェアではなかった．このことから両アルゴリズムで Gray となったアプリは高い確率でマルウェアの亜種であるといえる．一方，どちらかのアルゴリズムだけで Gray となったアプリは 52 種類あり，そのうち 29 種類はマルウェアではなかった．両アルゴリズムの場合に比べて割合は低いが，どちらかのアルゴリズムだけで Gray となったアプリにもマルウェアも含まれており，技術者が解析すべき検体を絞り込むという機能を果たしているといえる．

一方，Ivory となった科名で 101 種類のマルウェアのうち 48 種類は既知のマルウェアの亜種であり，完全に亜種を検出できたとはいえない結果であった．しかしあるマルウェアが既知のマルウェアの亜種なのか，それとも新しい科名が与えられるべき新種のマルウェアなのかを判断するのは解析者であり，同時に解析者は本研究の提案者でもある．したがって解析者が Ivory となったマルウェアは既知のマルウェアと異なるのだから新しい科名を与えるべきであると判断すれば，Ivory に既知のマルウェアの亜種はなくなり，完全に亜種を検出できたことになる．事実，表 4 や表 5 において，Adrd や Geinimi は亜種の定義ファイルでの包含度が高くなっているが，FakePlayer.a や KungFu.a は他の亜種の定義ファイルでの包含度は低い．これらに別の科名を与えたならば，この評価は変わってしまう．

本研究では解析者はマルウェアを命名するにあたって，そのマルウェアが一般的に何と呼ばれているか，アンチウイルス製品でどのような名前で検出されるのかを考慮した．実験の結果のための命名はしていないが，命名にあたっての判断に解析者の主観が入る余地はある．

結論としては，

- マルウェアから抽出した特徴である 163 種類の定義ファイルで，バイナリが異なる 1,978 種類のマルウェアを検出することができた．
- 両アルゴリズムで Gray となったアプリは高い確率で

既知のマルウェアの亜種である．

□ どちらかのアルゴリズムだけで Gray となったアプリは解析すべきである．

という結果から，マルウェアの検出や解析すべきアプリを見つけることはできたと我々は評価する．

5. 関連研究

Shabtai らは機械学習によりアプリの動作からマルウェアを検出する方法 [3] を提案している．また Burguera らはクラウドを利用したマルウェアを検出する方法 [4] を提案している．我々の研究とは異なりこれらの方法は動的解析であるため，マルウェアのコードが実行されなければ検出できない．一方，コードが異なる場合でもマルウェアとしての動作があるならば，未知のマルウェアを検出できる可能性がある．

澤谷らは動的解析を行いマルウェアを検出するときのシグネチャをする方法 [5] を提案している．これは動的解析に対する提案であり我々が提案する静的解析による方法とは異なるが，シグネチャを自動生成するという考え方は検討できるかもしれない．

Pouik らは Android のアプリにおいて，Normalized Compression Distance（NCD）を用いて類似度を算出する方法 [6] を提案している．この方法も我々の提案と同じく静的解析を行っているが，類似度の算出を NCD で行う点は我々の提案とは異なる．

本研究の提案は我々が過去に提案した方法 [2] の検討課題に対応したものであり，我々が過去に提案した Microsoft Windows のマルウェアを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方法 [7] を Android におけるマルウェア検出に応用したものである．

6. 今後の課題

本研究で提案した方法で対応できなかった種類のマルウェアは別の方法で対応することが考えられる．複数の方法を組み合わせて，検出洩れが生じないようにする必要がある．元々，本研究の提案は Windows のマルウェアを分類する方法 [7] を応用したものであるから，ネイティブコードに対応する方法を本研究とは別に実装することは可能だと思われる．

我々の過去の研究 [2] でも言及された定義ファイルの作成の困難さについては，まだ解析者の技術力や経験に依存する部分も多く完全に解決されたとはいえない．表 2 より，誤って Gray となった合計 32 種類のうちの多くは科名 Kiser によるものであった．特定の科名の定義ファイルが誤検出率を引き上げているので，定義ファイルの作成を工夫することで誤検出を減らすことが期待できる．そのためには定義ファイルの作成のためのツールを充実させ，解析者の技術力への依存を減らす必要がある．文献 [5] のような自動化の関連研究も参考にできる．

閾値の算出において，検体が亜種であるのか別種であるのかが関わっているが，この亜種・別種の判断は解析者が行っており，結果として閾値の決定は解析者の判断に依存することになる．また 2011 年 9 月の時点で誤って Gray とされたアプリが大量にあったため，それ以降は閾値の算出方法を改めて閾値を高めた．しかし誤検出を減らすために行った改変が，かえって検出できない m マルウェア検体を増加させてしまったかもしれない．この閾値の算出方法も検討の余地がある．

定義ファイルやアプリの数も文献 [2] のときに比べて増えているので，それらを管理して複数の方法を組み合わせて検査を行うシステムを構築する必要もある．最終的には検体をシステムに投入したときに自動的にシステムが複数の方法で解析を行いその結果を返すようにする予定である．本研究で提案した方法をそのシステムの一部としたいと我々は考えている．

参考文献

[1] Chen, X., Dirro, T., Greve, P., Li, H., Paget, F., Pogulievsky, V., Schmugar, C., Shah, J., Sherstobitoff, R., Sommer, D., Sun, B., Szor, P. and Wosotowsky, A.: McAfee Threats Report: Fourth Quarter 2012, McAfee Labs (online), available from ⟨http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2012.pdf⟩ (accessed 2013-04-01).

[2] 岩本一樹，和崎克己：制御フロー解析による Android マルウェア検出方法の提案，コンピュータセキュリティシンポジウム 2011 論文集，Vol. 2011, No. 3, pp. 714–719 (2011).

[3] Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C. and Weiss, Y.: Andromaly”: a behavioral malware detection framework for android devices, *Journal of Intelligent Information Systems*, Vol. 38, No. 1, pp. 161–190（オンライン），入手先 ⟨http://www.springerlink.com/index/10.1007/s10844-010-0148-x⟩ (2012).

[4] Burguera, I., Zurutuza, U. and Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for Android, *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, New York, NY, USA, ACM, pp. 15–26 (online), DOI: 10.1145/2046614.2046619 (2011).

[5] 澤谷雪子, 川端秀明, 磯原隆将, 竹森敬祐, 窪田　歩：Android マルウェアの挙動に基づく検知ルール自動化生成手法，暗号と情報セキュリティシンポジウム（SCIS 2012）(2012).

[6] Pouik and G0rfi3ld: Similarities for Fun & Profit, Phrack Inc. (online), available from ⟨http://www.phrack.org/issues.html?issue=68&id=15#article⟩ (accessed 2013-04-01).

[7] 岩本一樹，和崎克己：静的解析により抽出された API 推移に基づくマルウェアの分類，情報処理学会論文誌，Vol. 54, No. 3, pp. 1199–1210 (2013).