Android OS を対象とした仮想マルウェア解析環境の構築

M2013SC012 中野祐輔
指導教員：河野浩之

1　はじめに

世界の携帯端末市場では半数以上のシェアを Android OS が占めている．その中で Android 端末を狙った遠隔操作, 情報漏洩などのマルウェアによる被害が増えており社会問題となっている．被害を減らすためのマルウェア解析手段として, 静的解析と動的解析の手法がある．マルウェアの挙動を細かく解析していくためには静的解析が望ましいが, 時間と高度な知識が必要となる．一方で, 誰でも短時間でマルウェアを解析するには動的解析が有効となる．

本研究では, 動的解析を用いて Android 端末を狙うマルウェアのリアルタイム解析環境をエミュレータ上で構築する．エミュレータは, Common Open Research Emulator(以下 CORE) を用いて仮想ネットワークを構築し, DNS, HTTP, SMTP の疑似環境を提供する．構築した解析環境において, Android マルウェアを起動させた時点での通信の挙動を観測していく．観測するためのマルウェア検体として, Android Malware GENOME PROJECT[6] から提供された 1260 検体のマルウェアを所有したデータセット, contagio mobile[3] から入手した Android マルウェアを用いる．また, Google Play や一般サイトから入手するアプリケーションも利用する．そして, 解析環境上で検体を起動させることで, Android マルウェアの通信を観測しながら特徴を分析し, 考察する．

以下, 2 節で関連研究を説明する．3 節で解析環境の概要の説明, 4 節に Android マルウェアの観測方法, 5 節に観測結果の分析, 最後に 6 節でまとめる．

2　動的解析環境構築に関する研究

本節では, 本研究に関連する既存の研究を説明する．Android マルウェアによる問題を解決するための手法として, 西本は動的解析における Android における端末情報の取得検知手法の提案 [2] で提案している．アプリケーションフレームワークの中にログ出力するメソッドを挿入して, アプリケーションが API を通して情報を取得する際のログを記録して検査する手法を提案されている．実行時に出力されるログを実行終了時に解析するため, リアルタイム性に欠ける課題がある．しかし, ログメソッドは Android 内部の動作も観測できる利点があるため, 本研究では同様にログを観測できる DDMS を用いる．

また, 金井は動的解析による Android マルウェアの DNS 通信の観測 [4] で, Android マルウェアの解析をするための動的解析環境を構築して環境内でマルウェアを実行して通信の様子を観測している．結果, 名前解決や特徴的な DNS 通信を確認している．しかし, エミュレータであると相手に検知され, 挙動を示さない検体が数多く存在する結果となった．解決策として, Android の実機を用いて動的解析する方法が挙げられる．

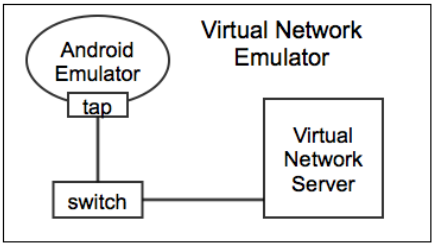

図 1　解析環境のシステムアーキテクチャ

本研究は既存研究の欠点を解決するためリアルタイム動的解析環境を構築する．ネットワークエミュレータ CORE を用いて疑似サーバを提供する．実ネットワークは繋がず, 独自のネットワーク環境を構築して Android エミュレータに提供する．Android エミュレータ上でマルウェアを実行し, 実行時の通信挙動に着目して解析する手法を提案する．

3　Android 端末による解析環境の概要

3 節では, 本研究で構築する解析環境, 技術について説明する．

3.1　解析環境のシステムアーキテクチャ

解析環境を構築するため, Android エミュレータとネットワークエミュレータの接続を既存研究である “ネットワークエミュレータ GINE の Android への組み込み”[1] を参考とし, 疑似サーバ提供方法は “ネットワークエミュレータ GINE を用いたマルウェア挙動解析環境の構築”[5] を参考にして Android マルウェアの解析環境を構築する．構築する為のホスト OS は Ubuntu13.04 64bit 版を使用する．Android エミュレータは Android Developers で提供されている Android SDK を用いてエミュレータを起動させる．Android OS は, Android SDK 付属の Android 4.0 を使用する．本研究で提案するシステムの構成を図 1 に示す．

3.2　Android エミュレータのネットワーク構成

SDK を利用して Android エミュレータを起動した場合のネットワーク構成を記述する．Android エミュレータは独自のネットワークアドレスを持ち, 仮想ルータが管理するネットワークアドレスは 10.0.2.x/24 である．本研究で用いるネットワーク構成は, 以下の割り当てられたネットワークアドレスが割り当てられている．それらのネットワーク構成を図 2 に示す．エミュレートされたデバイスのネットワーク制限として, IGMP やマルチキャストをサポートしておらず, 物理ネットワークにアクセスできない．そのため, PC システム全体を対象としたオープンソースのエミュレータである QEMU を用いて外部ネット

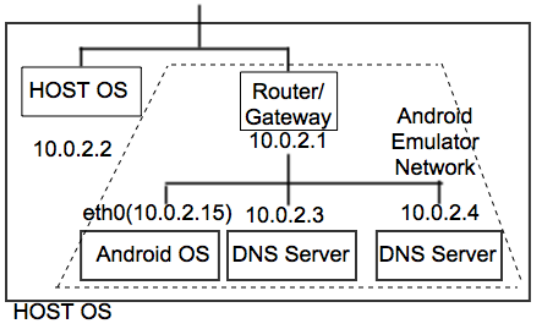

図 2　エミュレータのネットワーク構成

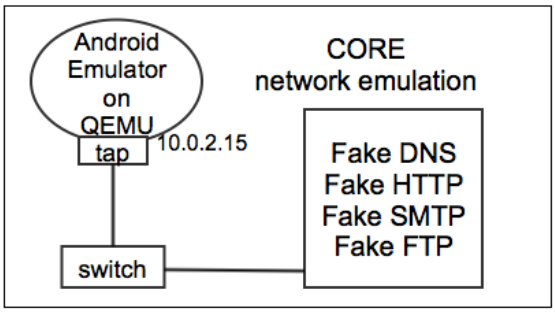

図 3　Android OS と CORE の接続

ワークに接続させる.

3.3　CORE と Android OS の接続

QEMU は, Android エミュレータを用いる際にゲスト OS として起動しており, tap 機能を用いることにより外部と通信するための設定ができる. tap 機能は, ゲスト OS と他の OS を繋ぐ tap でバイスを作成する機能であり, 他のホストと直接通信できるようになる. 本研究では, 外部ネットワークとして CORE で構築した仮想ネットワークと Android エミュレータを接続させることを目的とする. CORE は, ネットワークエミュレータであり, オブジェクトを用いて仮想ネットワークを構築することができるエミュレータである. そして, Android エミュレータで tap 機能を用いる場合は, オプションコマンドを用いて起動させなければならない.

エミュレータ起動時のコマンド

avd ”エミュレータ名” -qemu -net user, vlan=0 -net nic, vlan=1 -net tap, vlan=1, ifname=tap0

- -qemu QUEM を用いる場合のオプション
- -net user ユーザモードネットワークで起動するためのオプション
- -net nic 仮想ネットワークカードを作成するためのオプション
- -net tap tap デバイスを指定した OS に接続するためのオプション

以上のオプションコマンドを用いることで, QEMU と CORE を接続させて Android エミュレータを起動させることができる. CORE と Android エミュレータを接続させた場合の構成図を図 3 に示す.

CORE 上では Link Layer の RJ45 のオブジェクトを用いて QEMU と CORE を接続させる. tap デバイスを仮想スイッチに接続させることで CORE と QEMU 仮想ホストが接続させて QEMU 仮想ホストを独立させることができる.

3.4　疑似サーバの構築手法

本節では, 仮想インターネットで独自に提供する擬似サーバについて記述する. 擬似サーバ設置理由として, Android マルウェアは, 起動時に各サーバにリクエストを送り, 応答がない場合は起動を停止する可能性がある. そのため, リクエストに対して応答を返すサーバが必要となる. 本研究で設置する擬似サーバは, DNS, HTTP, SMTP である. 疑似 DNS サーバは NSD を用いて, 疑似 HTTP サーバはプログラムを組み, 疑似 SMTP サーバは Postfix を用いて疑似サーバを実現させる. それぞれの擬似サーバの概要を以下に記述する.

- 疑似 DNS サーバ

疑似 DNS サーバの設置には, BIND ではなく NSD を用いて仮想ネットワーク上に DNS ネームサーバを設置する. NSD は DNS コンテンツサーバの機能だけ提供しており, DNS キャッシュサーバの機能も持ち合わせる BIND よりもメモリの使用量を減らすことができ, 最低限の動作をさせることが可能となる. NSD の設定では, どのようなドメインでも同じサーバに誘導させるため, 設定ファイルにワイルドカードを記述する.

zone ファイルの記述例

* IN A 誘導先サーバ

- 疑似 HTTP サーバ

マルウェアからのリクエストに対してマルウェアにレスポンスする HTTP サーバを仮想ネットワーク上に稼働させる. リクエスト内容の特定は困難であるため, 常に同じレスポンスを返す HTTP サーバを awk で構築した. /inet/tcp を用いて常に通信を待ち続けるクライアントサーバとして稼働させる. そして, HTTP 要求に対してどのようなリクエストにも”HTTP/1.1 200 OK”と応答させる. 応答を返すことが目的のため, 存在しないファイルでも同じ応答を返すように設定した. そのため, ネットワーク通信の負荷を軽減することができる.

- 疑似 SMTP サーバ

仮想ネットワーク上に Postfix を用いて模倣 SMTP サーバを設置する. Android マルウェアによるメール中継を受け取ることを目的としてサーバを構築する. 複数のアドレスを同じ宛先に送信させるため, postfix の aliase に正規表現で記述する. しかし, aliase は直接正規表現を記述することはできないため, aliase-maps で別のファイルである aliases.regexp に正規表現を記述する.

正規表現の記述例

/(.*)/ /dummy/Maildir/

また, mydestination の値にワイルドカードを記述し, 全てのドメインがメールを配送されるように設定した.

4 動的解析によるマルウェアの挙動解析

本節では, 構築した環境で, Android マルウェア検体を用いて Android マルウェア起動時の通信の挙動を観測し, 解析する. マルウェアの挙動を観測する方法として, WireShark, Android SDK 付属ツールである Dalvik Debug Monitor Service(以下 DDMS) を用いた. WireShark では, Android エミュレータと外部との通信の流れを観測する. DDMS は, コマンドラインからデバッグできるツールである. 本研究では, ログの参照方法として用いる. また, Android マルウェアを評価するための検体は, Android Malware GENOME PROJECT, contagio mobile で提供されている Android マルウェアを用いて評価していく. Android Malware GENOME PROJECT は, Android プラットフォームに着目して 1,200 以上の Android マルウェアの検体を収集している. また, contagio mobile は, モバイルマルウェアを共有するために提供されているサイトであり, 200 検体以上のマルウェア検体をダウンロードすることができる.

4.1 実験目的

本研究においての実験目的は, Android マルウェア検体を用いて, 構築した解析環境において挙動を観測できるかどうかを検証する. 解析環境上で Android マルウェアを起動させた場合に, 特徴的な挙動が生じるかどうかの把握, 正規アプリケーションとそのリパッケージマルウェアの差異の検証. 更に, Android マルウェアの DNS 通信の特徴, HTTP 通信の特徴を検証することを目的とする. そこで, 実験 1 として, 本研究で構築した解析環境が正常に解析することが可能であるかを, 実ネットワークと接続した場合と本研究で構築した解析環境を比較して, 構築した解析環境の有効性を示す. 続いて実験 2 は, 幾つかの Android マルウェアアプリケーションを解析環境で起動したときの挙動などの特徴を記述する. 実験 3 では, 正規アプリケーションの挙動とそのリパッケージマルウェアを比較して違いを検証する. 最後に, 314 検体の Android マルウェアを起動させ, 通信挙動の特徴を考察する.

4.2 実験 1: 解析環境の有効性の検証

実験 1 では, 構築した解析環境において実ネットワークに接続されている環境と本研究で独自に提供している擬似サーバと比較して, 同等の観測結果が得ることができるのかを評価する. 評価検体は, contagio mobile から入手した”SpamSoldier-SMSBotnet”を用いた. 本研究で構築した解析環境では , DNS の名前解決を観測でき, 名前解決先に 1 度目の HTTP リクエストを観測することができる結果となった. そのため, 2 回以上の HTTP 通信の返答を繰り返す事で悪意のある反応を示す Android マルウェアには対応することができない. しかし, DNS 通信の悪意のある名前解決先の検出 , 1 度目の HTTP 通信による異常な挙動を観測することが可能であるため, 構築した解析環境での解析は場合に応じて有効であると言える.

4.3 実験 2: Android マルウェアの特徴的な挙動の観測

実験 2 では, Android マルウェアアプリケーションを解析環境上で起動し, その挙動の特徴を把握することができるか観測していく. 用いる検体は, contagio mobile から提供された, ”Enesoluty-fakeav-JapaneseSpyware” と呼ばれる Android マルウェア検体で構築した解析環境上で挙動を観測する. また, Android マルウェアアプリケーションの起動時の挙動を, WireShark と DDMS を使用して挙動を観測し分析する. WireShark での挙動結果を図 4 に記載する.

```
DNS, 72, standard query 0x2aba A app-roid.com
HTTP, 94, POST /app/rv.php?id=”Android ID” HTTP/1.1
```

図 4 WireShark における挙動結果

HTTP リクエストの中身によると, id=以降は Android-ID の固定値を示す値となっている. また, リクエストの中身を詳しく見てみると, Content-Type が multipart/form-data であるため, ファイルを取得して list.tex をどっかにアップデートしている様子が推測される. WireShark だけではなく, DDMS も用いて確認する. Log.d でログを確認して得た結果を図 5 に記載する.

```
12-10 : D/xxx(700): start
12-10 : D/xxx(700): mailaddress get!/mnt/sdcard
12-10 : D/xxx(700): org.apache.http.client.
                    ClientProtocolException
12-10: D/xxx(700): post end
```

図 5 DDMS による結果

Android マルウェア起動時のログによると, メールアドレスをゲットしたと推測される結果が記述されている. WireShark だけではなく, DDMS も同時に用いる事でより細かい解析結果を得ることが可能になった. 結果的に, 用いた Android マルウェアは通常のアプリケーションを装い, 裏でアドレスを盗み, 攻撃元に送信するマルウェアであることが解析できた. よって, マルウェア特有の特徴を構築した解析環境上で観測できると結果となった.

4.4 実験 3: 正規アプリケーションとリパッケージマルウェアの比較

実験 3 では, 正規のアプリケーションとそのリパッケージマルウェアを解析環境上で起動し, どのような違いを観測することができるのかを検証して考察する. 用いる検体は, contagio mobile から入手した FlashPlayer のリパッケージマルウェアである FakeFlashPlayer11 を用いた. 図 6, 7 で通信の挙動観測結果を記載する.

”adobe.com” の名前解決だけでなく, ”kimokama.com” の名前解決がされており, adobe とは関係ないサイトに問い合わせしている様子が観測できる.

```
DNS 76 Standard query 0xe858 A www.kimokama.com
DNS 92 Standard query response 0xe858 0xe858 A 10.0.2.2
DNS 74 Standard query 0xdfd3 A www.adobe.com
DNS 92 Standard query response 0xdfd3 A 10.0.2.2
```

図 6 DNS 通信の挙動

```
HTTP 450 /multimedia/en_us/get/flash/flash_animation.wef
HTTP  50 HTTP/1.x 200 OK (text/html)
HTTP 310 GET /data.php?action=cmd6oonline=0000....
HTTP  50 HTTP/1.x 200 OK (text/html)
HTTP 310 GET /data.php?action=cmd6oonline=0000....
HTTP  50 HTTP/1.x 200 OK (text/html)
```

図 7 HTTP 通信の挙動

HTTP による要求は, ”adobe.com”に向けて一回 HTTP 要求し, 後は”kimokama.com”へ HTTP 要求を続けた. 正規のホームページから取得した, install-flash-player.apk は HTTPS 通信であるため, 解析環境では挙動を詳しく観測することができなかった. しかし, 通常のアプリケーションでは, 暗号化を行うのに対し, Android マルウェアは基本的に暗号化をせず HTTP で通信することが考えられる. そして, 悪質なアプリケーションでは正常なアプリケーションと比べて挙動が増えることが推測される結果となった.

4.5 Android マルウェアの通信特徴分析

本節では, Android Malware GENOME PROJECT で提供された 1260 個 49 種類の Android マルウェアの中から, 比較的最新であるトロイの木馬マルウェアの特徴を分析していく. Android マルウェアが起動した時点での DNS, HTTP 通信の挙動を観測し, 考察する. 用いたマルウェア検体は以下の表 1 にまとめる.

表 1 分析する検体名, 検体数

検体名	検体数
AnserverBot	187
ADRD	22
Bgserv	9
DroidKungFu4	96

結果的に, AnserverBot は, 少なくとも 1 個の DNS の問い合わせがなされており, 一番多かった DNS の問い合わせは 5 個, 平均 2.83 個の名前解決を観測することができた. 表 2 に検体の名前解決先の一例を記述する.

ドメイン”www.sosceo.com”について, シマンテックのセキュリティレスポンスにて, 特定の Android アプリケーションにバンドルされている広告ライブラリであると記述されている. 2014 年 5 月時点でリスクインパクトが high となっている事から, 悪性ドメインだと推測できる. また, ドメイン”b4.cookier.co.cc”は, 様々な Android マルウェアに関する研究する際に公開されているドメインと類似しており, 悪性ドメインである可能性が高いということがわかる.

その他にも, ADRD では, 平均 1.42 個, Bgserv は平均 2.7 個, DroidKungFu4 は平均 2.3 個の名前解決を観測できた. 全体では, 1 つのアプリケーションで, 平均して 2.3 個の DNS による名前解決がなされている結果となった. 名前解決先のドメインもそれぞれ悪性と予想されるドメインであることが見受けられるため, 過度な量の名前解決がなされていたらマルウェアであると疑う必要があると考えられる.

表 2 AnserverBot の問い合わせ先

名前解決先ドメイン	目的 (推測)
www.sosceo.com	悪性 (広告)
b4.cookier.co.cc	悪性
r.domob.cn	広告
www.jiemai-tech.com	アプリケーション配信サイト

5 おわりに

本稿では, 擬似 DNS, HTTP, SMTP サーバを提供して Android マルウェアを動的解析する環境を構築した. そして擬似サーバを提供した解析環境の有効性を確認するため, 実ネットワークと接続した場合の挙動と比較して評価した. さらに, 構築した解析環境上で Android マルウェアを起動し, 特徴的な DNS, HTTP の通信を観測した. 構築した解析環境上で特徴的なマルウェアの挙動を把握することが可能となることで, 短期間, 安全性の高い環境でマルウェアを解析することが可能となる.

今後の課題として, Android マルウェアの判断基準の特定, 今回実装しなかった他の擬似サーバの実装など解析環境の改善が挙げられる.

参考文献

[1] 馬場 隆章, 後藤 邦夫, “ネットワークエミュレータ GINE への Android の組み込み”, FIT2010 第 9 回情報科学技術フォーラム, 第 4 分冊, pp. 189-190 (2010).

[2] 西本 祐揮, 堀 良彰, 櫻井 幸一, “動的解析を用いた Android における端末情報の取得検知手法”, 情報処理学会研究報告, 火の国情報シンポジウム (2012).

[3] Mila Parkour, “contagio mobile”, http://contagiominidump.blogspot.jp/ (accessed 2015-01-11).

[4] 金井 文宏, 吉岡 克成, 松本 勉, “動的解析による Android マルウェアの DNS 通信の観測”, 情報通信システムセキュリティ研究会, Vol. 112, pp. 31-36 (2013).

[5] M. Yasuaki and K. Goto, “Design and implementation of malware analysis using network emulator gine,” in Computer Security Symposium 2012(CSS2012), Vol. 2012, No. 3, pp. 114-121 (2012).

[6] Y. Zhou and X. Jiang, “Dissecting Android Malware:Characterization and Evolution”, In Proc. of the IEEE Symposium on Security and Privacy (2012).