UNIVERSIDADE FEDERAL DE PERNAMBUCO

Centro de Informática

Graduação em Ciência da Computação

UM ESTUDO SOBRE OS PROBLEMAS DE SEGURANÇA NO SISTEMA OPERACIONAL ANDROID

Dyego Felipe Oliveira da Penha

Trabalho de Graduação

Recife, Julho de 2015

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Centro de Informática

Graduação em Ciência da Computação

UM ESTUDO SOBRE OS PROBLEMAS DE SEGURANÇA NO SISTEMA OPERACIONAL ANDROID

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção de grau de Bacharel em Ciência da Computação.

Orientador: Paulo André da Silva Gonçalves (pasg@cin.ufpe.br)

Aluno: Dyego Felipe Oliveira da Penha (dfop@cin.ufpe.br)

Recife, Julho de 2015

AGRADECIMENTOS

A Deus, que tem me direcionado durante toda a minha vida e me capacitou a realizar este trabalho.

À minha família, que sempre me depositou confiança, construiu a pessoa que sou hoje e tem estado comigo em todas as horas.

Aos meus amigos, destacando os do Centro de Informática, por todos os desafios e projetos que realizamos durante o curso com muita dedicação e trabalho em equipe.

Ao professor Paulo Gonçalves, por me orientar neste trabalho e direcionar ao melhor caminho.

ÍNDICE DE FIGURAS

LISTA DE ACRÔNIMOS

JDK Java Development Kit.

SDK Software Development Kit.

APK Android Package.

API Application Programming Interface.

IDC International Data Corporation.

DVM Dalvik Virtual Machine.

JVM Java Virtual Machine.

IPC Inter Process Communication.

DAC Discretionary Access Control.

UID Unique Identifier.

GID Group Identifier.

OEM Original Equipment Manufacturer.

OTA Over The Air.

PMS Package Management System.

NDK Native Development Kit.

MAC Mandatory Access Control.

NCD Normalized Compression Distance.

RESUMO

Esta monografia apresenta um breve estudo sobre os problemas de segurança no sistema operacional Android e como se prevenir ou defender destes problemas. Será abordado, também, os fundamentos da plataforma Android, com o intuito de fornecer ao leitor uma base de conhecimento para melhor compreensão das vulnerabilidades e ameaças que atingem o sistema operacional. Além disso, será visto as melhorias de segurança desenvolvidas pela Android a cada nova versão, as principais técnicas de análise e detecção de malwares, e boas práticas de defesa tanto para o desenvolvedor como para o usuário de aplicativos. Ademais, será proposto um aplicativo, denominado Check-Up, que pretende oferecer aos usuários Android mais segurança em seus dispositivos. Este aplicativo é responsável por realizar uma análise no dispositivo identificando as vulnerabilidades do sistema, mostrar o relatório da análise para o usuário, e por fim, sugerir correções de segurança. Este trabalho apresentará o funcionamento e as interfaces do aplicativo, bem como testes que visam atestar sua qualidade.

Palavras-chave: Android, Segurança, Vulnerabilidades, Malwares, Defesas

ABSTRACT

This paper presents a brief study on the security issues on the Android operating system and how to prevent or defend these problems. Will be addressed, too, the fundamentals of the Android platform, in order to provide the reader with a knowledge base for better understanding of vulnerabilities and threats that affect the operating system. It will also be seen security improvements developed by Android with each new release, the main techniques of analysis and detection of malware, and good defense practices for both the developer as to the application user. In addition, an application called Check-Up, which aims to offer Android users more security on your devices will be proposed. This application is responsible for performing an analysis on the device identifying system vulnerabilities, showing the scan log to the user, and finally suggest security fixes. This paper presents the operation and application interfaces as well as tests aimed at vouch for its quality.

Keywords: Android, Security, Vulnerabilities, Malwares, Defenses

SUMÁRIO

INTRODUÇÃO

O mercado de smartphones em todo o mundo cresceu 27,7% no ano de 2014, com um total de vendas de 1,3 bilhões de unidades, de acordo com os dados do International Data Corporation (IDC) [1]. Ainda em 2014, o Android ultrapassou a marca de um bilhão de unidades, um marco significativo em si, pois o número total de aparelhos com o sistema operacional Android superou o volume total de smartphones vendidos em 2013. Atualmente, a plataforma Android domina o mercado global de dispositivos móveis, com uma quota de aproximadamente 80% [1]. Mas, enquanto os valores de mercado do Android continuam crescendo, também cresce as críticas sobre os protocolos de segurança da empresa [2]. Além dos problemas de segurança com os aplicativos autorizados pela Google para venda na Google Playstore, o relatório da empresa de segurança cibernética OPSWAT sugeriu que cerca de um terço dos aplicativos disponíveis para venda em lojas de aplicativos de terceiros contêm *malware* (ou aplicativos maliciosos) [2].

De acordo com os últimos dados em 2013 da empresa de segurança F-Secure, 97 por cento de todo o *malware* móvel tem como alvo dispositivos Android [3]. Em 2012, esse número foi de 79 por cento. O que é pior, o número total de assinaturas de *malware* está em ascensão. Em 2012, foram identificados 238 ameaças na empresa móvel Android. Agora, esse número é de 804. Essas estatísticas, juntamente com a preocupação constante entre os clientes corporativos que nenhuma solução de segurança única chega perto de resolver problemas do mundo móvel, fazem com que várias empresas se preocupem com a segurança no Android.

A plataforma Android, liderada e mantida pela empresa Google, é um projeto desenvolvido sob licença de código aberto, pela Android Open Source Project (AOSP), e por esta razão sofre o problema de fragmentação [4], existem várias versões do Android no mercado, mesmo em dispositivos atuais. Os fabricantes costumam fazer suas próprias alterações no Android, para que eles pudessem estar por trás da liberação de referência atual do Google. Além disso, operadoras e fabricantes podem não atualizar a versão Android de seus dispositivos quando o Google faz, ou levar meses ou até mesmo anos para fazê-lo.

O Android permite que qualquer pessoa desenvolva as próprias aplicações e distribua livremente. Mas quando aberto oferece várias conveniências para desenvolvedores e usuários, como também aumenta o risco de segurança. Devido à falta de desenvolvimento de aplicativos de controle efetivo, o usuário geralmente baixa e instala aplicativos maliciosos escrito por hackers de software. Isto irá resultar em algumas ou todas as características do telefone móvel não funcionar adequadamente [5].

Por causa da grande influência desse sistema operacional móvel no mercado global e dos riscos que têm apresentado, este trabalho estuda as vulnerabilidades de segurança e ameaças que cercam o Android. Com o objetivo de apresentar técnicas e ferramentas de segurança ao desenvolvedor, como também medidas de proteção para o usuário desta plataforma.

1.1 Objetivos

Com o atual desenvolvimento das tecnologias móveis, e o domínio global do sistema operacional Android em tais dispositivos, o objetivo deste trabalho é realizar um estudo sobre os problemas de segurança e riscos envolvidos, aos quais este sistema operacional está sujeito.

Será apresentado um breve estudo sobre os fundamentos do Android, as suas vulnerabilidades e ameaças que o cercam. Também será discutido questões de segurança, com o intuito de fornecer ao desenvolvedor de aplicativos uma base para implementar e distribuir aplicativos de forma mais segura. Aos usuários, será apresentado os riscos que eles estão sujeitos ao instalar aplicativos Android e como eles podem se proteger.

Por fim, será desenvolvido um aplicativo para realizar testes a respeito das questões de segurança discutidas, e então, as conclusões obtidas serão apresentadas.

1.2 Estrutura do Documento

Este documento foi estruturado em capítulos bem definidos, para que haja um melhor entendimento por parte do leitor. Estes capítulos constroem uma base a respeito do tema deste trabalho que se desenvolvem até chegar ao objetivo real proposto, discutido na seção anterior. Deste modo, espera-se que todos os conceitos sejam entendidos por completo e o leitor compreenda todo o contexto em que o trabalho se encontra.

O primeiro capítulo introduz o trabalho, mostrando os principais conceitos e dando uma visão geral dos capítulos seguintes, bem como o objetivo principal.

O segundo capítulo discursa a respeito dos fundamentos do sistema operacional Android e suas primícias para segurança. Neste capítulo, será abordado a arquitetura do Android e suas camadas, os componentes de um aplicativo, e como esses componentes se comunicam. Alguns princípios que possuem relação com a segurança no Android também serão discutidos, como o conceito de Sandboxing, permissões, e a loja de aplicativos oficial Google Play.

O terceiro capítulo tem como foco apresentar todos os problemas de segurança que afetam o Android. Será discutido aqui, as ameaças mais comuns ao qual a plataforma está exposta. O conceito de fragmentação, um dos maiores problemas que este sistema operacional possui e que gera inúmeras vulnerabilidades. O risco causado por executar aplicativos utilizando código nativo. Principais *malwares* conhecidos do estado-da-arte. As alternativas que estes *malwares* utilizam para se infiltrar no dispositivo e também as técnicas que eles usam para se ocultar das ferramentas de análise e detecção de *malwares*.

No quarto capítulo, serão apresentadas as melhorias de segurança que o Android tem desenvolvido a cada nova versão, e as melhorias de segurança ao sistema operacional por parte de terceiros. Serão discutidas aqui, as principais ferramentas do estado-da-arte para realizar análises e detecções de vulnerabilidades e aplicativos maliciosos.

O quinto capítulo fornece ao leitor, conceitos de segurança tanto do ponto de vista do programador, como do usuário de aplicativos Android. Ao desenvolvedor, boas práticas e ferramentas, com o objetivo de desenvolver uma aplicação mais segura e esquivar as vulnerabilidades apresentadas anteriormente. Ao usuário, vários passos, com o fim de se prevenir contra ameaças e diminuir o risco de expor as suas informações confidenciais.

No sexto capítulo, será apresentado o aplicativo desenvolvido e os resultados dos testes que foram realizados.

Finalmente, serão apresentadas as conclusões, indicando as considerações finais a respeito do trabalho discutido.

CAPÍTULO 2

FUNDAMENTOS

Esta seção, faz uma breve descrição da arquitetura do Android, estrutura do aplicativo, componentes, além de outros conceitos importantes da plataforma, a fim de que o leitor adquira uma base de conhecimento sobre o Android para melhor compreensão das questões de segurança que serão abordadas no Capítulo 3.

2.1 Arquitetura

O sistema operacional Android é uma pilha de componentes de software, que está dividido em cinco seções e quatro camadas principais [6], como mostrado abaixo no diagrama da arquitetura.

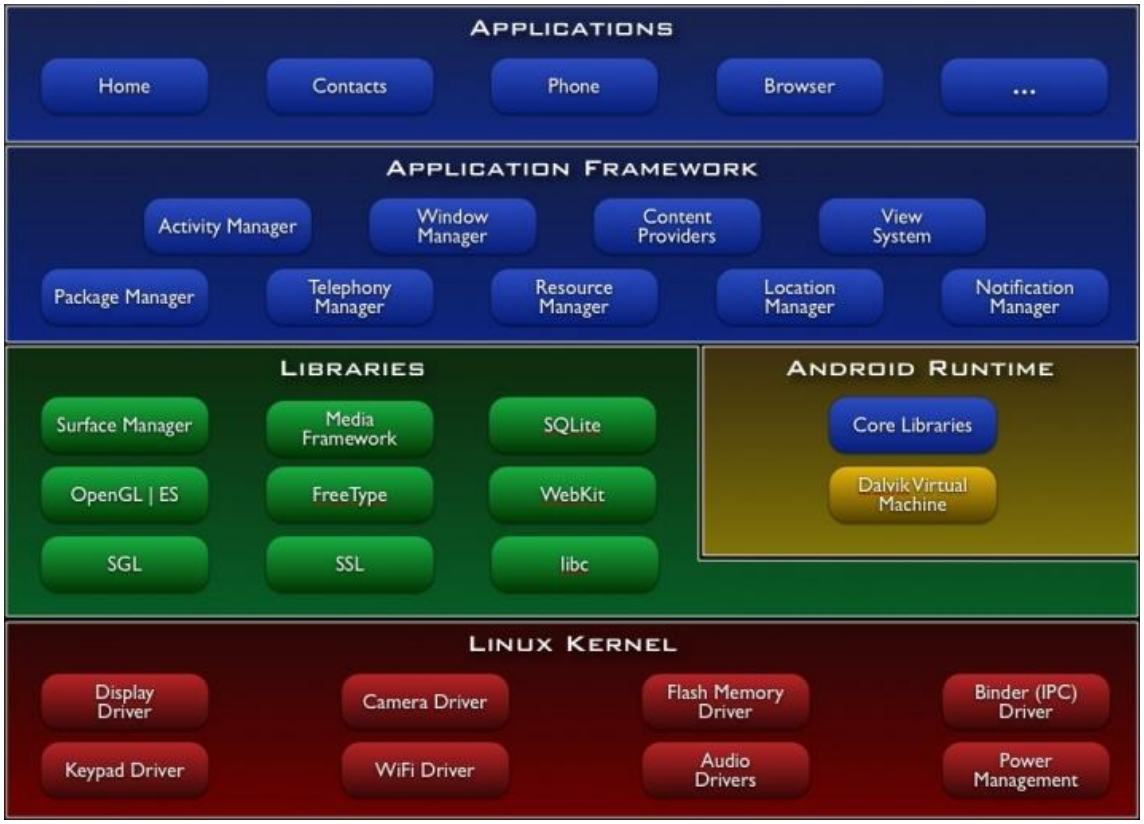

Figura 1. Arquitetura do Sistema [7].

- **Linux Kernel**: O núcleo do sistema operacional é construído em cima do *Kernel* (ou núcleo) do Linux. Isso fornece a funcionalidade básica do sistema, como gerenciamento de processos, gerenciamento de memória, gerenciamento de dispositivos como câmera, teclado, display, etc. Além disso, o *Kernel* lida com todas as coisas que o Linux é realmente bom em fazer, como redes e uma vasta gama de drivers de dispositivos, que levam a abstração da interface com hardware periférico.

- **Libraries**: No topo do *Kernel* do Linux, existe um conjunto de bibliotecas de código aberto, incluindo Web Browser Engine WebKit, a biblioteca libc, banco de dados SQLite que é um repositório útil para armazenamento e compartilhamento de dados de aplicativos, bibliotecas para iniciar e gravar

áudio e vídeo, bibliotecas SSL responsáveis pela segurança da Internet, entre outras bibliotecas.

- **Android Runtime**: Esta seção fornece um componente chave chamado Dalvik Virtual Machine (DVM), que é uma espécie de Java Virtual Machine (JVM) especialmente projetado e otimizado para o Android. O Dalvik VM faz uso de recursos do núcleo Linux como gerenciamento de memória e multi-threading, que é intrínseca na linguagem Java. O Dalvik VM permite que cada aplicativo possa ser executado em seu próprio processo, com a sua própria instância da máquina virtual Dalvik. O Android Runtime também fornece um conjunto de bibliotecas do núcleo que permitem que os desenvolvedores de aplicativos Android possam escrever aplicativos usando a linguagem de programação Java, no entanto, o código nativo e bibliotecas compartilhadas são desenvolvidas em C/C++.

- **Application Framework**: A camada Application Framework fornece vários serviços de nível superior para aplicações na forma de classes Java. Os desenvolvedores de aplicativos têm permissão para fazer uso desses serviços em suas aplicações. Como por exemplo o Notification Manager, que fornece ao desenvolvedor o serviço de notificações que pode ser implementado em seu aplicativo.

- **Applications**: Todos os aplicativos Android se encontram nesta camada superior. O desenvolvedor vai escrever aplicativos que serão instalados apenas nesta camada. Exemplos de tais aplicações são: Contatos, Browser, Jogos, e as mais variadas aplicações.

O Android protege as funcionalidades sensíveis, tais como telefonia, GPS, rede, gerenciamento de energia, rádio, mídia e outros serviços do sistema, com o modelo baseado em permissão, que será visto na subseção 2.6.

Para proteger o acesso não autorizado ou modificações, a partição do sistema Android é do tipo somente leitura. Além disso, uma parte do sistema de arquivos como o cache do aplicativo e SDCard estão protegidos com os privilégios apropriados, para prevenir a sua adulteração pelo adversário quando o dispositivo está ligado à área de trabalho através do USB [8].

2.2 Estrutura do aplicativo

O aplicativo Android é empacotado em um APK, um arquivo zip que consiste de vários arquivos e pastas, conforme mostrado na Figura 2 abaixo.

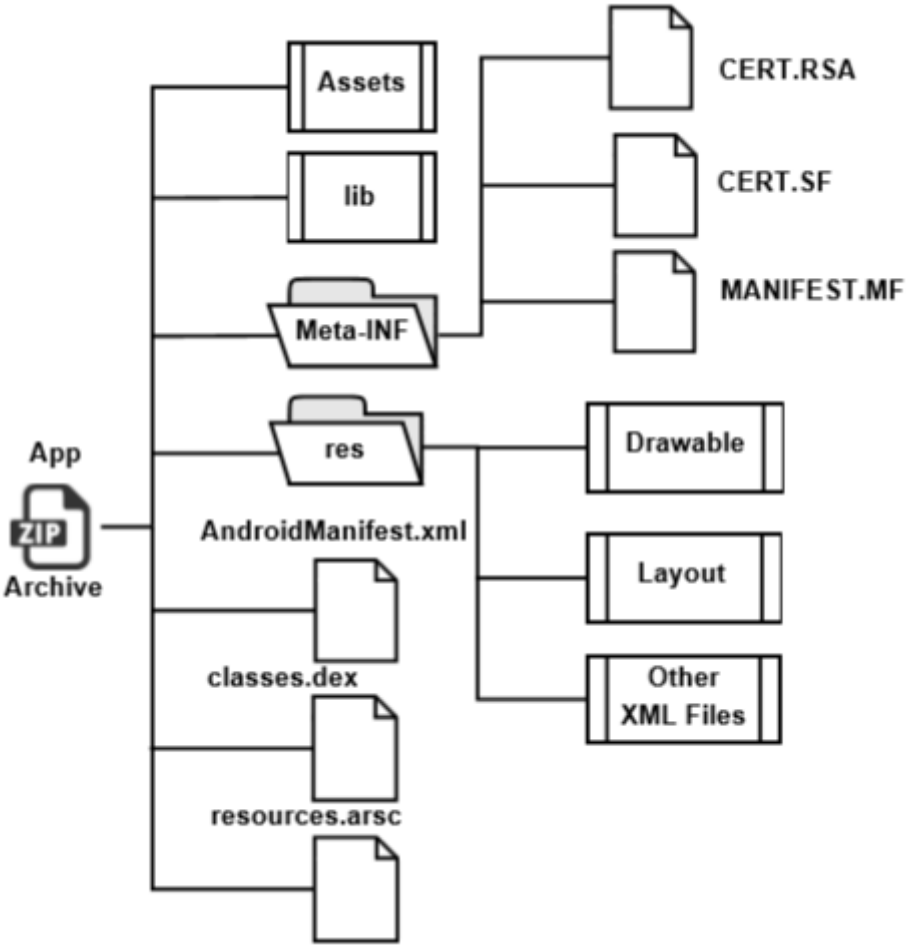

Figura 2. Estrutura do Android PacKage (APK) [9].

- **AndroidManifest.xml**: Antes de o sistema Android poder iniciar um componente do aplicativo, o sistema deve saber que o componente existe lendo o arquivo AndroidManifest.xml. Devem ser declarados todos os componentes do aplicativo neste arquivo, que deve estar na raiz do diretório do projeto. O manifesto faz uma série de coisas, além de declarar os componentes do aplicativo, tais como:
    - Identificar quaisquer permissões de usuário que o aplicativo requer, como acesso à Internet ou acesso de leitura para os contatos do usuário.

- Declarar a versão mínima do Android exigida pela API do aplicativo, baseado em quais APIs o aplicativo usa.
- Declarar recursos de hardware e software utilizados ou exigidos pelo aplicativo, como uma câmera, serviços Bluetooth, ou uma tela multi touch.
- Bibliotecas da API que a aplicação necessita para serem ligadas, como por exemplo a biblioteca da Google Maps.

- **Res**: Essa pasta é responsável por armazenar os recursos do aplicativo, compilados para o binário, tais como:
  - **Drawable:** Todas as imagens utilizadas pelo aplicativo são armazenadas nesta pasta.
  - **Layout:** Todos os arquivos XML de layout se encontram nesta pasta, tais como: layouts de menu, layouts das telas do aplicativo. Os layouts são a representação gráfica da interface com o usuário.
  - **Other XML Files:** Nesta pasta, é possível definir quaisquer arquivos XML que o desenvolvedor do aplicativo julgar necessário, tais como: Strings, Números, Cores, Animações e arquivos XML para armazenar as preferências do menu de configurações do aplicativo.

- **Assets**: Nesta pasta, estão contidos recursos não compilados. Tais como imagens, texturas, áudio, vídeo, ou qualquer arquivo que o desenvolvedor deseje preservar e que pode ser lido como um Stream de Bytes.
- **Lib**: Contém as bibliotecas que serão ligadas ao aplicativo, conforme descrito no arquivo AndroidManifest.xml.
- **META-INF**: Armazena a assinatura do desenvolvedor do aplicativo certificado, para verificar a identidade do desenvolvedor de aplicativos de terceiros.
- **Classes.dex**: O arquivo executável classes.dex armazena o bytecode Dalvik a ser executado na máquina virtual Dalvik.

- **Resources.arsc**: Os recursos que foram compilados para o binário, ou seja, a pasta res, tem os seus nomes e identificadores mapeados no arquivo resources.arsc.

Como mencionado anteriormente, os aplicativos Android são desenvolvidos em Java. O processo de desenvolvimento é ilustrado na Figura 3 abaixo. O código Java compilado gera um número de arquivos .class (bytecode Java intermediário das classes definidas no projeto). Usando a ferramenta Dx, arquivos .class são fundidos em um único arquivo executável Dalvik (.dex). O arquivo .dex armazena o bytecode Dalvik a ser executado em uma DVM para acelerar a execução.

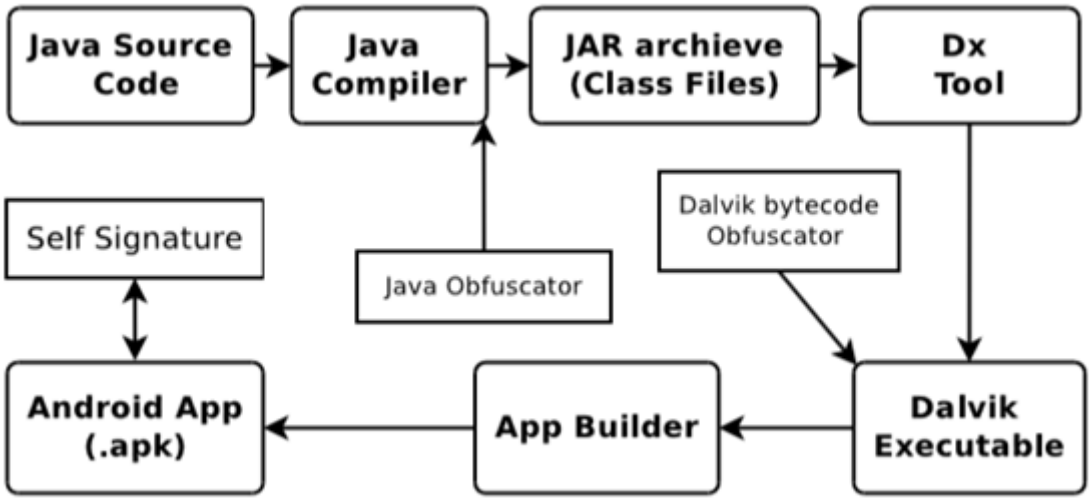

Figura 3. Processo de construção do app [9].

2.3 Componentes

Os componentes são os blocos de construção essenciais de um aplicativo Android. Cada componente é um ponto diferente, através do qual o sistema pode entrar em seu aplicativo. Nem todos os componentes são os pontos de entrada reais para o usuário e alguns dependem um do outro, mas cada um existe como uma

entidade própria e desempenha um papel específico em um bloco de construção exclusivo que ajuda a definir o comportamento geral do seu aplicativo [10].

Estes componentes são fracamente acoplados pelo manifesto (arquivo AndroidManifest.xml) do aplicativo que descreve cada componente da aplicação e como eles interagem [11]. Existem quatro tipos diferentes de componentes de aplicativos, e cada tipo tem uma finalidade distinta.

Uma aplicação Android é composta de um ou mais dos componentes discutidos abaixo:

- **Activities**: Uma *Activity* (ou atividade), representa uma única tela com uma interface de usuário. Por exemplo, um aplicativo de e-mail pode ter uma *Activity* que mostra uma lista de novos e-mails, uma outra *Activity* para compor um e-mail, e uma outra *Activity* para a leitura de e-mails. Embora as *Activities* trabalhem em conjunto para formar uma experiência de usuário coesa no aplicativo de e-mail, cada uma é independente das outras. Como tal, uma aplicação diferente pode ser iniciada em qualquer uma destas *Activities* (se o aplicativo de email permitir). Por exemplo, um aplicativo de câmera pode iniciar a *Activity* no aplicativo de e-mail que escreve um novo e-mail, para que o usuário possa compartilhar uma foto.

- **Services**: Um *Service* (ou serviço), é um componente que roda em segundo plano para executar operações de longa execução ou executar trabalhos para processos remotos. Um *Service* não fornece uma interface de usuário. Por exemplo, um *Service* pode reproduzir música em segundo plano enquanto o usuário estiver em um aplicativo diferente, ou pode buscar dados sobre a rede sem bloquear a interação do usuário com uma atividade. Outro componente, como uma atividade, pode iniciar o *Service* e deixá-lo correr ou ligar para ele, a fim de interagir com ele.

- **Content Providers**: Um *Content Provider* (ou provedor de conteúdo), gerência um conjunto compartilhado de dados de aplicativos. Você pode armazenar os dados no sistema de arquivos, um banco de dados SQLite, na web, ou qualquer outro local de armazenamento persistente que seu aplicativo pode acessar. Através do *Content Provider*, outros aplicativos podem consultar ou mesmo modificar os dados (se o *Content Provider* permite que ele o faça). Por exemplo, o sistema Android fornece um *Content Provider* que gerência informações de contato do usuário. Como tal, qualquer aplicativo com as permissões adequadas pode consultar parte do provedor de conteúdo (como ContactsContract.Data) para ler e gravar informações sobre uma determinada pessoa. Os *Content Providers* também são úteis para a leitura e escrita de dados que é privado para o seu aplicativo e não compartilhado. Por exemplo, o aplicativo de exemplo Note Pad usa um *Content Provider* para salvar notas. O armazenamento de dados é acessível por meio das URIs definidas no aplicativo.

- **Broadcast Receivers**: Um *Broadcast Receiver* (ou receptor de transmissão), é um componente que responde aos anúncios de transmissão de todo o sistema. Muitas transmissões originam-se do sistema, por exemplo, uma transmissão anunciando que a tela se desligou, a bateria está fraca, ou uma imagem foi capturada. Aplicativos também podem iniciar as transmissões, por exemplo, para permitir que outros aplicativos saibam que alguns dados foram baixados para o dispositivo e está disponível para eles usarem. Apesar de *Broadcast Receivers* não apresentarem uma interface de usuário, eles podem criar uma notificação na barra de status para alertar o usuário quando ocorre um evento de transmissão. Mais comumente, no entanto, um *Broadcast Receiver* é apenas uma "porta de entrada" para outros componentes e destina-se a fazer uma quantidade mínima de trabalho. Por

exemplo, pode iniciar um serviço para executar algum trabalho com base no evento.

Um aspecto único do projeto do sistema Android é que qualquer aplicativo pode iniciar o componente de outro aplicativo. Por exemplo, se você quiser que o usuário tire uma foto com a câmera do dispositivo, há provavelmente um outro aplicativo que faz isso e seu aplicativo pode usá-lo, em vez de desenvolver uma *Activity* para capturar uma foto. Você não precisa incorporar ou mesmo ligar o código do aplicativo da câmera. Em vez disso, você pode simplesmente iniciar a *Activity* no aplicativo da câmera que captura uma foto. Quando concluída, a foto é devolvida ao seu aplicativo para que você possa usá-la. Para o usuário, parece como se a câmera é realmente uma parte de seu aplicativo.

Como o sistema funciona com cada aplicativo em um processo separado com permissões de arquivo que restringem o acesso a outros aplicativos, o seu aplicativo não pode ativar diretamente um componente de outro aplicativo. O sistema Android, no entanto, pode. Então, para ativar um componente de outro aplicativo, você deve entregar uma mensagem ao sistema que especifica a sua intenção de iniciar um componente particular. O sistema então ativa o componente para você.

2.4 Comunicação entre os componentes

A plataforma Android suporta a comunicação entre processos (ou Inter-Process Communication - IPC), para a comunicação entre os componentes [12]. Uma fundação para este IPC é o Binder, um dispositivo do *Kernel* especifico do Android que permite uma comunicação eficiente e segura.

A maneira de se comunicar com os componentes não são conhecidas no tempo de desenvolvimento, são mensagens, que podem incluir dados arbitrários,

chamados *Intents* (ou intenções). Uma intenção é uma descrição abstrata de uma operação a ser executada na plataforma [13]. Por exemplo, uma intenção de iniciar uma nova *Activity* ou *Service*, ou se comunicar com serviços em segundo plano. Uma vantagem desta técnica é que um aplicativo cliente não está mais vinculado a um programa específico, mas pode acessar qualquer serviço possível para a necessidade específica.

O Android tem dois métodos básicos de aplicação de segurança para a comunicação entre os componentes. Em primeiro lugar, os aplicativos são executados como processos Linux com seus próprios IDs de usuário e, portanto, são separados um do outro. Dessa forma, uma vulnerabilidade em uma aplicação não afeta outras aplicações. Em contraste com o Java, a máquina virtual não é uma barreira de segurança, pois o *kernel* Linux assume a tarefa de separação de processos. Desde que o Android fornece mecanismos de IPC, que precisam ser protegidos, um segundo mecanismo de aplicação entra em jogo. O Android implementa um monitor de referência para mediar o acesso aos componentes da aplicação com base em rótulos de permissão [14].

Quando um componente inicia o IPC, o monitor de referência olha para os rótulos de permissão atribuídos ao container do seu aplicativo. Se a etiqueta de permissão exigida pelo componente a ser iniciado está declarada no seu aplicativo, o sistema permite que o IPC inicie o componente requisitado. Se a etiqueta não for encontrada, o estabelecimento do IPC é recusado mesmo que os componentes sejam uma parte do mesmo aplicativo. O desenvolvedor atribui etiquetas de permissão através do arquivo AndroidManifest citado anteriormente na subseção 2.2. O desenvolvedor define a política de segurança do aplicativo, visto que a atribuição de permissão para os componentes em uma aplicação especifica uma política de acesso para proteger seus recursos.

2.5 Sandbox

O Android foi construído como um sistema operacional móvel seguro, com o motivo de proteger os dados do usuário, os aplicativos dos desenvolvedores, o dispositivo, e a rede [8]. No entanto, a segurança depende da vontade do desenvolvedor e capacidade de aderir às melhores práticas de desenvolvimento. Além disso, o usuário deve estar ciente do efeito que um aplicativo pode ter sobre a segurança dos dados e dispositivo. Soluções *anti-malware* não têm direitos suficientes para executar verificações de *malware* agressivas devido ao modelo de segurança do sistema operacional. Por exemplo, aplicativos *anti-malware* têm uma varredura, capacidade de monitoramento e sistema de arquivos restrito no dispositivo.

O *Kernel* Android implementa o Linux Discretionary Access Control (DAC). Cada processo do aplicativo é protegido com um identificador único (UID) dentro de um *Sandbox* isolado. O *Sandbox* restringe os aplicativos ou serviços do sistema de interferir em outro aplicativo. O Android protege o acesso à rede através da implementação do recurso Paranoid Network Security, que é utilizado para controlar o Wi-Fi, Bluetooth e acesso à Internet dentro dos grupos de ID [15]. Se uma aplicação tem a permissão para um recurso de rede (por exemplo, Bluetooth), é atribuído ao processo do aplicativo o ID correspondente de acesso à rede. Portanto, além de UID, pode ser atribuído a um processo um ou mais ID de grupo (GID). O *Sandbox* de aplicativos Android é ilustrado na Figura 4.

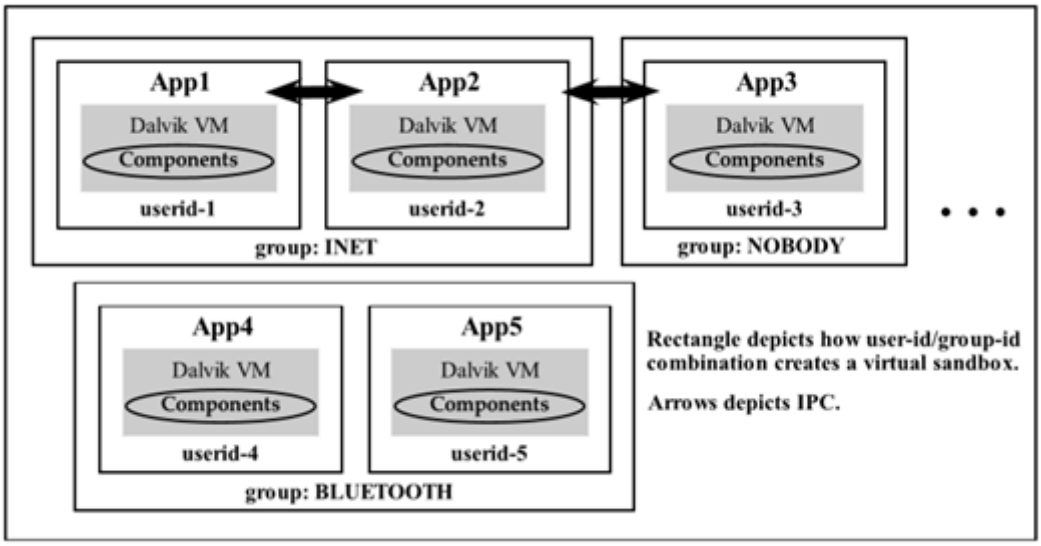

Figura 4. Apps Android com Sandbox no nível do Kernel [8].

Um aplicativo deve conter um certificado PKI assinado com a chave de desenvolvedor (ver Figura 3). A assinatura do aplicativo é o ponto de confiança entre o Google e os desenvolvedores de terceiros para garantir a integridade do aplicativo e a reputação do desenvolvedor. O procedimento de assinatura do aplicativo coloca um aplicativo em um *Sandbox* isolado atribuindo-lhe um UID exclusivo [9]. Se o certificado de um aplicativo "A" combina com o de um aplicativo "B" já instalado no dispositivo, o Android atribui o mesmo UID (ou seja, o mesmo *Sandbox*) para as aplicações A e B, que lhes permite compartilhar os seus arquivos privados e as permissões definidas no manifesto. Este compartilhamento não intencional pode ser explorado pelos criadores de *malware*. É aconselhável que os desenvolvedores mantenham privado os seus certificados para evitar a sua utilização abusiva.

2.6 Permissões

Para restringir um aplicativo de acessar as funcionalidades sensíveis, tais como telefonia, rede de contatos, SMS, SDCard e localização GPS, o Android fornece um modelo de segurança baseado em permissões no âmbito da aplicação. O desenvolvedor deve declarar as permissões necessárias usando a tag <uses-permissions> no arquivo AndroidManifest.xml como discutido anteriormente. O Android controla os aplicativos individuais para mitigar os efeitos indesejáveis sobre os aplicativos do sistema ou aplicativos de desenvolvedores de terceiros dentro da *Sandbox*. Estas restrições são impostas sobre o processo no momento de instalação. As permissões do Android estão divididas em quatro níveis de proteção como veremos a seguir [16]:

- **Normal**: Estas permissões têm um risco mínimo para o usuário, sistema, ou o dispositivo. Permissões normais são concedidas por padrão no momento de instalação.

- **Dangerous** (ou Perigosas): Essas permissões abrangem o grupo de alto risco, devido a sua capacidade de acessar os dados privados e sensores importantes do dispositivo. Um usuário deve aceitar a instalação de permissões perigosas no momento de instalação.

- **Signature** (ou Assinatura): Essas permissões são concedidas apenas se o aplicativo requerente está assinado com o mesmo desenvolvedor certificado do aplicativo que declarou as permissões. Elas são concedidas automaticamente no momento de instalação. Permissões de assinatura estão disponíveis com os aplicativos do sistema.

- **SignatureOrSystem** (ou AssinaturaOuSistema): Essas permissões são concedidas se o aplicativo que está solicitando é assinado com o mesmo

certificado da imagem do sistema Android ou que é assinado com o mesmo certificado como o aplicativo que declarou a permissão. A permissão *SignatureOrSystem* é utilizada para certas situações especiais, em vários fornecedores têm aplicações construídas em uma imagem do sistema e precisam compartilhar características específicas, explicitamente, porque elas estão sendo construídas em conjunto. Essas permissões são concedidas automaticamente no momento da instalação.

Permissões Android são *Coarse-grained* (ou de granularidade grossa). Por exemplo, a permissão INTERNET não tem a capacidade de restringir o acesso a um específico URL. A permissão READ_PHONE_STATE permite que um aplicativo identifique se o dispositivo está tocando ou está em espera. Ao mesmo tempo, as permissões também permitem que o aplicativo leia as informações sensíveis, tais como identificadores de dispositivo. Permissões como WRITE_SETTINGS, CAMERA são globalmente definidas, portanto, viola o princípio do acesso de privilégio mínimo. Acesso a WRITE_CONTACTS ou WRITE_SMS não implica o acesso a READ_CONTACTS ou permissões READ_SMS [9]. Assim, as permissões não são hierárquicas e elas devem ser solicitadas separadamente pelo desenvolvedor. No momento da instalação, o usuário é obrigado a conceder todas as permissões ou negar a instalação do aplicativo. Daí as permissões perigosas não podem ser evitadas no momento de instalação. Além disso, os usuários não conseguem diferenciar entre a necessidade e seu mau uso imperativo que possam expor as suas informações para fins de exploração [17].

2.7 Loja Google Play

A Google desencoraja os usuários a instalarem aplicativos fora da loja oficial Google Playstore, para impedir que os usuários instalem aplicativos de qualquer

mercado de terceiros, devido à preocupação de segurança. No entanto, ele ainda permite a instalação de aplicativos a partir de outros mercados de terceiros.

Aplicativos de desenvolvedores de terceiros são disponibilizados a partir da loja oficial, onde a Google veta, se necessário, o aplicativo do desenvolvedor de terceiros com o Bouncer [18], que é uma ferramenta de análise dinâmica em um ambiente *Sandbox* com a finalidade de impedir qualquer *malware* de entrar na loja Google Play.

O Bouncer, é um mecanismo de segurança razoavelmente eficaz [19]. O Android tem a facilidade de execução de um serviço de verificação ao instalar aplicativos de outros lugares de mercado. A Google Play é capaz de desinstalar remotamente um aplicativo, se encontrar um comportamento malicioso [20]. No entanto, esta facilidade está disponível apenas para os dispositivos conectados à Internet.

Para publicar um aplicativo é necessário dois passos básicos, segundo a Figura 5, de preparação e publicação do aplicativo, que serão discutidas a seguir.

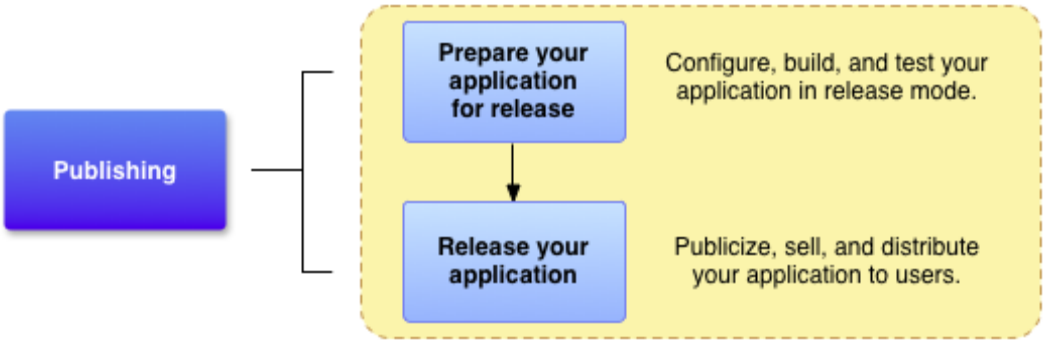

Figura 5. App Publishing [21].

A primeira etapa se refere a preparação do aplicativo para que este possa ser lançado na loja. Para esta etapa, o desenvolvedor deve efetuar cinco passos importantes, conforme a Figura 6 abaixo.

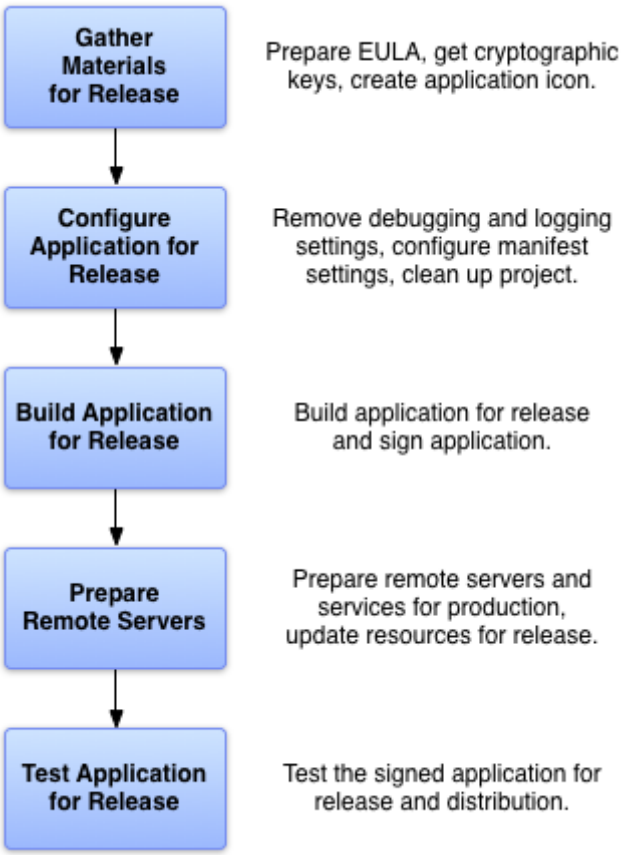

Figura 6. Preparing Application for Release [21].

- **Gather Materials for Release**: Obter os materiais para o lançamento, como o material necessário para marketing, fotos, e descrição do aplicativo. Existem diversos materiais opcionais que o desenvolvedor pode adicionar, contudo os requisitos mínimos são:
    - Preparar o EULA (*End-User License Agreement*), ou licença de software.
    - Obter as chaves de criptografia, necessário para assinar o aplicativo na loja.
    - Criar o ícone da aplicação.

- **Configure Application for Release**: Certifique-se de desativar o log e desativar a opção de depuração antes de construir o lançamento da sua aplicação. Você pode desativar o log, removendo as chamadas de métodos Log em seus arquivos de origem. Você pode desativar a depuração, removendo o “android: debuggable”, atributo da tag <application> em seu arquivo de manifesto, ou definindo o atributo “android: debuggable” para falso. Além disso, remova quaisquer arquivos de log ou arquivos de teste estáticos que foram criados em seu projeto. Limpe o projeto e certifique-se que está em conformidade com a estrutura de diretórios descrita em [22]. Deixar arquivos inutilizados em seu projeto pode impedir que o aplicativo compile e fazer com que a aplicação se comporte de forma imprevisível.

- **Build Application for Release**: Após terminar de configurar seu aplicativo você pode construí-lo em um arquivo .apk que está assinado e otimizado. O JDK inclui as ferramentas para assinar o arquivo .apk (Keytool e Jarsigner); o Android SDK inclui as ferramentas para compilar e otimizar o arquivo .apk.

- **Prepare Remote Servers**: Se a sua aplicação depende de um servidor remoto, certifique-se de que o servidor é seguro e que ele está configurado para uso em produção. Isto é particularmente importante se você estiver implementando *In-app Billing* [23] em seu aplicativo e você está realizando a etapa de verificação de assinaturas em um servidor remoto. Além disso, se o seu aplicativo busca o conteúdo de um servidor remoto ou de um serviço em tempo real (como um feed de conteúdo), verifique se o conteúdo que você está oferecendo é atualizado e pronto para produção.

- **Test Application for Release**: Testar a versão de lançamento do seu aplicativo ajuda a garantir que seu aplicativo é executado corretamente em um dispositivo e sob condições de rede realistas. Idealmente, você deve testar

seu aplicativo em pelo menos um dispositivo do tamanho de um smartphone, e um dispositivo do tamanho de tablet, para verificar se os seus elementos de interface do usuário são dimensionados corretamente e que o desempenho do aplicativo e eficiência da bateria são aceitáveis.

Após esses passos serem concluídos, o aplicativo está pronto para ser publicado. Para isso, é necessário registrar uma conta de desenvolvedor Google Play, que por sua vez, requer uma taxa de 25 dólares. Quando o seu registro é verificado, o desenvolvedor pode acessar através da sua conta o *Developer Console* [24] (ver Figura 7), que é a casa para as suas operações de publicação de aplicativos e ferramentas no Google Play.

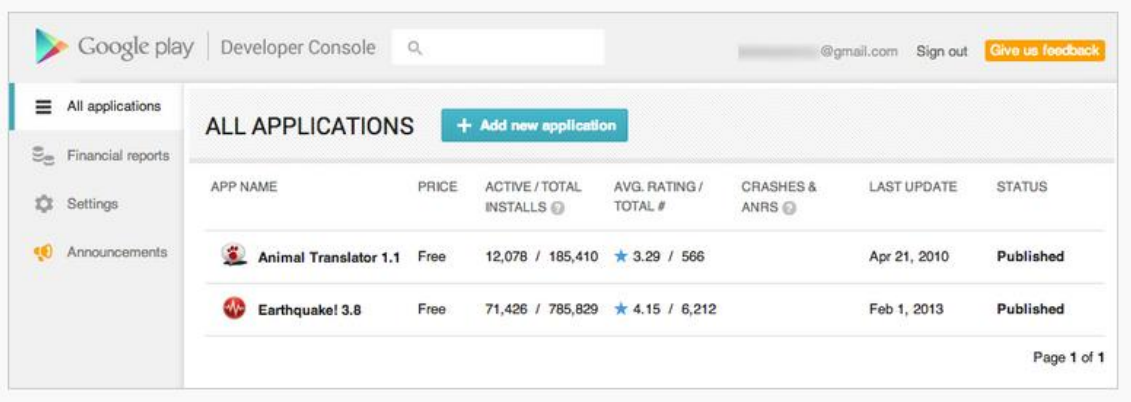

Figura 7. Developer Console [24].

CAPÍTULO 3

PROBLEMAS

Esta seção apresenta uma descrição sobre as questões de segurança do usuário e do dispositivo. Serão discutidas as principais ameaças, o conceito de fragmentação no Android, a vulnerabilidade de desenvolver um aplicativo utilizando o código nativo, os malwares conhecidos do estado da arte, como estes malwares se infiltram no sistema e a maneira com a qual eles esquivam as ferramentas de detecções.

3.1 Ameaças

A Android Open Source Project (AOSP) está comprometida com um sistema operacional seguro, mas, também é suscetível a ataques de engenharia social. Uma vez que o aplicativo é instalado, ele pode criar consequências indesejáveis para a segurança do dispositivo. A seguir, há uma lista de atividades maliciosas que foram relatadas ou podem ser utilizadas em todas as versões posteriores do Android.

- Ataques de escalas de privilégios foram alavancados, explorando vulnerabilidades do *Kernel* do Android, disponíveis publicamente para ganhar acesso à raiz do dispositivo [25]. Os componentes exportados do Android podem ser explorados para ganhar acesso às permissões perigosas.
- Vazamento de privacidade ou o roubo de informações pessoais, ocorre quando os usuários concedem permissões perigosas para aplicativos maliciosos e, sem saber, permitem o acesso a dados sensíveis que são vazados sem o conhecimento ou consentimento do usuário.

- As aplicações maliciosas também podem espionar os usuários, monitorando as chamadas de voz, SMS/MMS, mTANs (Números de autenticação de transações móveis) [26] bancárias, gravação de áudio/vídeo sem o conhecimento ou consentimento do usuário.
- As aplicações maliciosas podem ganhar dinheiro fazendo chamadas ou subscrever número de telefone em alguma promoção via SMS sem o conhecimento ou consentimento do usuário.
- Podem comprometer o dispositivo para funcionar como um Bot, e controla-lo remotamente através de um servidor, por meio do envio de vários comandos para realizar atividades maliciosas.
- Campanhas publicitárias agressivas podem motivar os usuários a baixar aplicativos potencialmente indesejados, ou aplicativos maliciosos [27].
- *Colluding Attack* (ou Ataques coletivos), acontece quando um conjunto de aplicativos, assinado com o mesmo certificado, é instalado em um dispositivo. Esses aplicativos irão compartilhar UID uns com os outros, e também qualquer permissão perigosa solicitada por um aplicativo será compartilhada pelo *malware*. Coletivamente, esses aplicativos irão executar atividades maliciosas, ao passo que, a sua funcionalidade individual é benigna. Por exemplo, um aplicativo com permissão READ_SMS pode ler as SMS e pedir ao parceiro do ataque coletivo com permissão INTERNET para exportar as informações confidenciais para um servidor remoto.
- O ataque de negação de serviço (também conhecido como *DoS Attack*), pode acontecer quando o aplicativo usa excessivamente a CPU, memória, bateria, ou recursos de largura de banda e com isso os usuários tem o desempenho prejudicado ao executarem funções normais do sistema.

3.2 Fragmentação

A Android Open Source Project (AOSP), liderado pelo Google, atualiza e mantém o código-fonte do Android. No entanto, uma atualização ou versão principal de distribuição de atualização é da responsabilidade dos fabricantes de equipamentos originais (OEMs) ou as operadoras de telefonia móvel. Muitas OEMs pegam versões desatualizadas do sistema operacional Android para fazer as suas modificações. Em alguns países, as operadoras de telefonia móvel personalizam o sistema operacional da OEM para atender às suas próprias necessidades. Essa cadeia de atualização leva meses antes do patch alcançar os usuários finais. Este fenômeno é chamado de fragmentação, onde diferentes versões do Android permanecem dispersas devido à indisponibilidade de atualizações. Especificamente, aparelhos com versões mais antigas e sem patch permanecem vulneráveis as ameaças conhecidas.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.9%
4.1.x	Jelly Bean	16	17.3%
4.2.x		17	19.4%
4.3		18	5.9%
4.4	KitKat	19	40.9%
5.0	Lollipop	21	3.3%

Data collected during a 7-day period ending on March 2, 2015.
Any versions with less than 0.1% distribution are not shown.

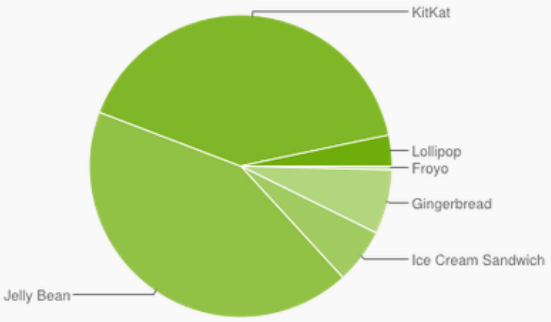

Figura 8. Distribuição do Android [28].

A Figura 8 acima apresenta a distribuição das versões do Android segundo a Google Playstore. É possível observar, que uma minoria utiliza a versão mais nova disponibilizada pela Google. Além disso, a partir de pesquisa realizada com aproximadamente 682.000 dispositivos, foram identificados cerca de 20.000 variações do sistema operacional Android instaladas nestes dispositivos [29]. Estas

pesquisas, na verdade, denotam apenas uma fração da realidade, visto que, atualmente o Android está instalado em bilhões de dispositivos.

Existem inúmeras vulnerabilidades que podem ser exploradas no Android, [30] apresenta uma lista com todas vulnerabilidades, explicitando o risco que elas causam e a maneira como podem ser exploradas por adversários, que foram registradas desde a versão mais antiga do Android até a versão mais recente. A Bluebox, recentemente, realizou uma análise com vários dispositivos Android de fabricantes distintos, com o intuito de avaliar os riscos de segurança que as versões modificadas podem apresentar [31]. A seguir, as vulnerabilidades mais comuns encontradas em alguns dispositivos:

- Modo de depuração USB ativado por padrão: O sistema roda com privilégios de usuário root.
- Aplicativo assinado com chave de teste da AOSP: Permite ao adversário facilmente criar um Malware, como por exemplo um Trojan.
- Backdoor de segurança pré-instalado: Também conhecido como pre-rooted, isto inclui o comando “su” instalado pela fábrica.
- Não inclui a loja Google Play: Requer o uso de lojas de terceiros para instalar aplicativos.
- Desabilita por padrão a configuração de segurança que protege a instalação de aplicativos a partir de fontes de terceiros mal-intencionados.
- Remove alguns recursos de segurança: como autorizar a conexão ADB no dispositivo.
- Aplicativos pré-instalados: o vendedor poderia modificar o aplicativo.

Atualizações do sistema operacional Android são mais frequentes em relação a atualizações de sistema operacional de Desktops. O Android lançou 29 atualizações estáveis de versão do sistema operacional, e atualiza desde o seu lançamento em Setembro de 2008 [32]. A atualização da Over The Air (OTA) altera significativamente

a versão existente modificando o grande número de arquivos através da plataforma, mantendo a integridade dos dados do usuário e aplicativos existentes [33]. Uma nova atualização de versão é facilitada através de um serviço chamado de Sistema de Gestão de Pacotes (PMS). [33] apresenta um estudo abrangente sobre as vulnerabilidades que por sua vez podem ser exploradas pelos autores de *malware* durante as atualizações de versão. Uma aplicação desenvolvida para a versão mais antiga pode ser explorada para usar a permissão perigosa introduzida na versão mais nova. Durante a atualização, o Android não verifica as permissões em anexo no aplicativo atualizado [33], comprometendo a segurança do dispositivo. Um grande número de arquivos são modificados durante uma atualização, e deve ser assegurado que as informações sensíveis do usuário permaneçam intactas, abstraindo a complexidade dos procedimentos da atualização.

3.3 Código nativo

O Android permite a execução de código nativo através de bibliotecas implementadas em C/C++ usando o Native Development Kit (NDK). O código nativo é executado fora da Dalvik VM, mas está em área restrita através da combinação UID/GID. No entanto, o código nativo tem o potencial para realizar escalonamento de privilégios, explorando vulnerabilidades da plataforma [34], que foi demonstrada por alguns ataques de *malware* que ocorreram recentemente [35].

3.4 Malwares

A Figura 9 ilustra a linha do tempo de algumas famílias notáveis de *malware* do Android durante o período de 2010 à 2013. Entre elas, SMS Trojans têm importante contribuição; alguns deles até mesmo infectaram o Google Playstore [35]. Um grande número de aplicações de *malware* exploraram ataques baseados no root, tais

como Rage-against-the-cage [36], Gingerbreak [37] e z4root [38] a fim de obter privilégios de superusuário para controlar o dispositivo. A vulnerabilidade mais recente explorada no Android é o ataque Master-key [39], que explora vulnerabilidades a partir da versão 1.6 até a 4.2.2 do Android.

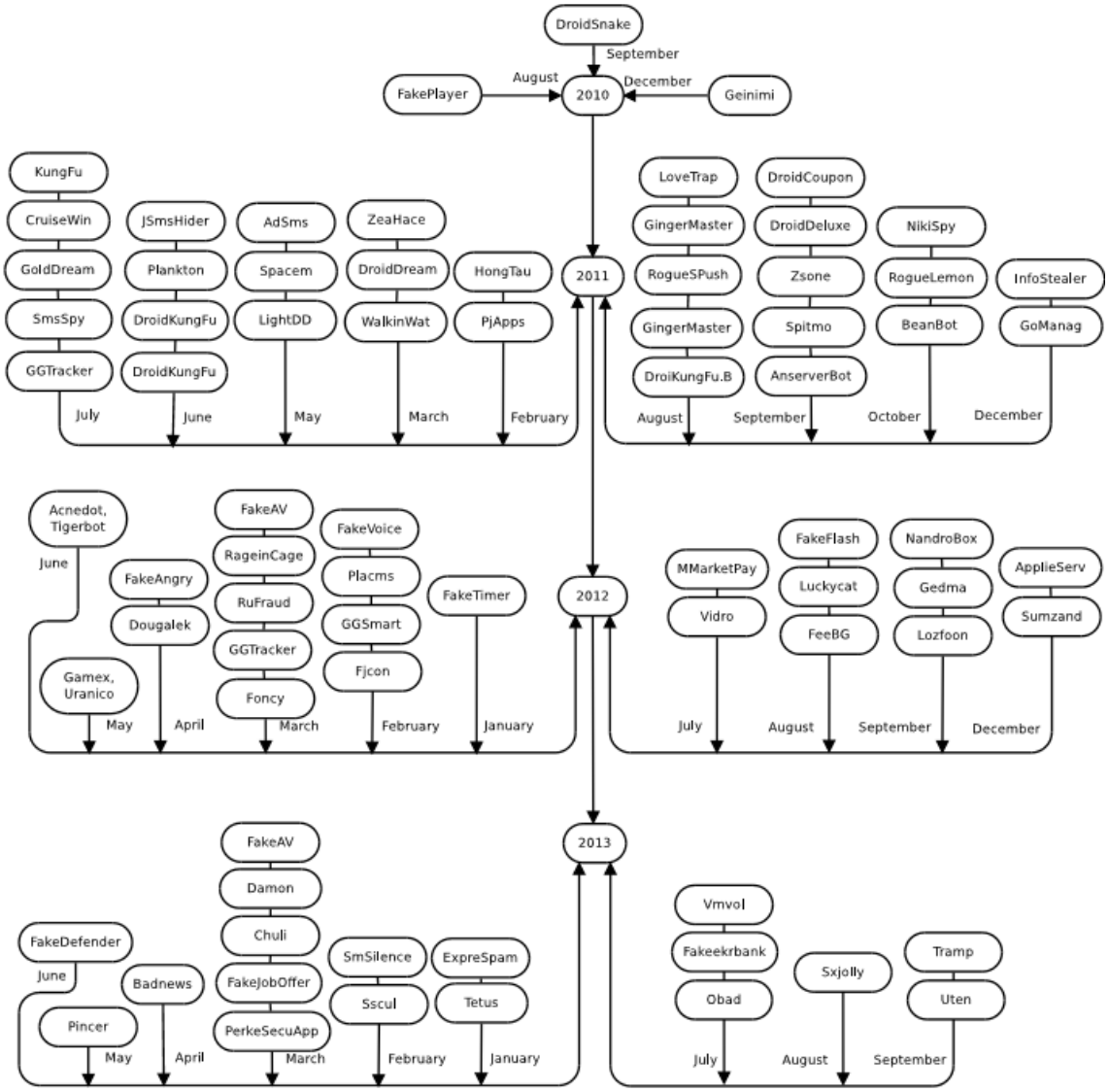

Figura 9. Cronologia das famílias de malwares Android [40].

A cada trimestre, as empresas de *anti-malware* relatam um aumento exponencial das novas famílias e variantes de *malware* existentes [41], [42]. Estas empresas diferem na aproximação da taxa de infecção de *malware* em dispositivos Android. Em particular, Lookout Inc. relatou a taxa de infecção global por *malware* de aproximadamente 2,61% de seus usuários [43]. Duas pesquisas independentes estimaram a taxa de infecção real.

- Em [44], o artigo utilizou o tráfego de DNS dos smartphones nos Estados Unidos e relatou infecção de 0,0009%.
- Em [45] foi desenvolvido o aplicativo Carat [46], para estimar as taxas de infecção por três conjuntos de dados de *malware* diferentes, relatando 0,26% e 0,28% para o conjunto de dados McAfee e Mobile *Sandbox* respectivamente. Portanto, a percepção atual de ameaça em Android e a taxa de infecção por *malware*, tem uma enorme variação registrada entre a propaganda *anti-malware* e os estudos independentes.

No parágrafo seguinte, será discutido os *malwares* classificados e suas características.

3.4.1 Trojan

Trojans se mascaram como aplicativos benignos, mas eles realizam atividades nocivas sem o consentimento ou conhecimento dos usuários. Trojans vazam as informações confidenciais do usuário, ou podem enganar o usuário e roubar informações sensíveis, como as senhas. Até o primeiro semestre de 2012, a maioria das variantes do Android pertenceu a várias famílias de SMS Trojan. Aplicativos SMS Trojan são capazes de enviar SMS para números de telefones sem o conhecimento ou consentimento do usuário incorrendo em perda financeira para o proprietário. Além disso, esses Trojans também vazam contatos, mensagens, números de IMEI/IMSI para os domínios de comando e controle. O FakeNetflix [47] se disfarça

como o popular aplicativo NetFlix, fazendo com que o usuário digite suas credenciais de login. FakePlayer [27], Zsone [41] e Android.Foney [48] são alguns Trojans Android notáveis que resultam em perda financeira para o usuário.

Em 2010, o usuário "09Droid" vendeu cerca de 40 aplicações diferentes de Mobile Banking na loja oficial da Google [49]. Infelizmente, nenhum destes aplicativos estavam filiados com os respectivos bancos.

Por conta do aumento das transações bancárias móveis, os autores de malware têm como alvo a autenticação desses serviços. Depois de capturar o nome de usuário e senha de contas alvo empregando ataques de engenharia social, os Trojans monitoram e roubam os mTANs (Números de autenticação de transações móveis) para silenciosamente completar transações [50].

3.4.2 Backdoor

O Backdoor permite que outros tipos de *malware* entrem silenciosamente no sistema facilitando-lhes o desvio dos procedimentos normais de segurança. Ele pode empregar *root exploits* (ou explorar vulnerabilidades no root) para obter o privilégio de superusuário e se esconder dos scanners *anti-malware*. Um exemplo de *root exploits* como o Rage-againt-the-cage [36], e Gingerbreak [37] ganham controle completo do dispositivo. Basebridge [35], KMin [35], Obad [51] são exemplos notáveis dos backdoors conhecidos.

3.4.3 Worm

O aplicativo conhecido como Worm pode criar cópias exatas ou similares de si mesmo e espalha-las através da rede ou mídia removível. Por exemplo, Worms Bluetooth podem explorar a funcionalidade Bluetooth e enviar cópias aos

dispositivos emparelhados. Android.Obad.OS [51] é um Worm Bluetooth bem conhecido.

3.4.4 Botnet

Aplicativos classificados como Botnet comprometem o dispositivo para criar um Bot, de modo que o dispositivo é controlado por um servidor remoto, chamado Botmaster, através de uma série de comandos. A rede de tais Bots é chamada de Botnet. Os comandos podem ser tão simples como o envio de informações privadas para um servidor-remoto ou complexas levando a ataques de negação de serviço (DoS Attack). Bot também pode incluir comandos para baixar códigos maliciosos automaticamente. Geinimi [35], Anserverbot [35], Beanbot [35] são botnets Android notáveis.

3.4.5 Spyware

O Spyware pode se apresentar como um bom utilitário, mas tem uma agenda escondida para monitorar clandestinamente contatos, mensagens, localização, mTANs bancários, entre outras coisas, que leva a consequências indesejáveis. Ele pode enviar as informações coletadas para o servidor remoto. Nickyspy [35], GPSSpy [41] são exemplos de aplicativos de Spyware.

3.4.6 Agressive Adware

Android fornece serviços de localização de granularidade grossa e fina. Algumas redes de anúncios de uso indevido afiliam-se a serviços de localização e enviam anúncios personalizados para o dispositivo do usuário para gerar renda. O Agressive Adware pode criar atalhos na tela inicial, roubar bookmarks, alterar as configurações do mecanismo de pesquisa padrão e forçar notificações desnecessárias para

dificultar o uso eficaz do dispositivo. Plankton [41] é um Agressive Adware conhecido.

3.4.7 Ransomware

O Ransomware pode bloquear o dispositivo do usuário para torná-lo inacessível até que um certo valor de resgate seja pago através do serviço de pagamento online. Por exemplo, FakeDefender.B [52] disfarçado como o *anti-malware* avast! [53], exibe os alertas de *malware* falsos para convencer o usuário a instalar este falso *malware*. Além disso, ele bloqueia o dispositivo e exige resgate para desbloquear o dispositivo.

3.5 Malwares: como eles se infiltram?

Nesta seção, é resumido os meios pelos quais os *malwares* se infiltram no Android.

3.5.1 Remontagem de aplicativos

Remontagem é um processo de desmontagem dos aplicativos de lojas populares de aplicativos Android, inserção de *Payload* (ou carga) do *malware*, e remontagem do aplicativo malicioso, que é distribuído através das lojas de aplicativos locais menos monitoradas. Um aplicativo pode ser remontado com as ferramentas de engenharia reversa existentes. O processo de remontagem é ilustrado na Figura 10.

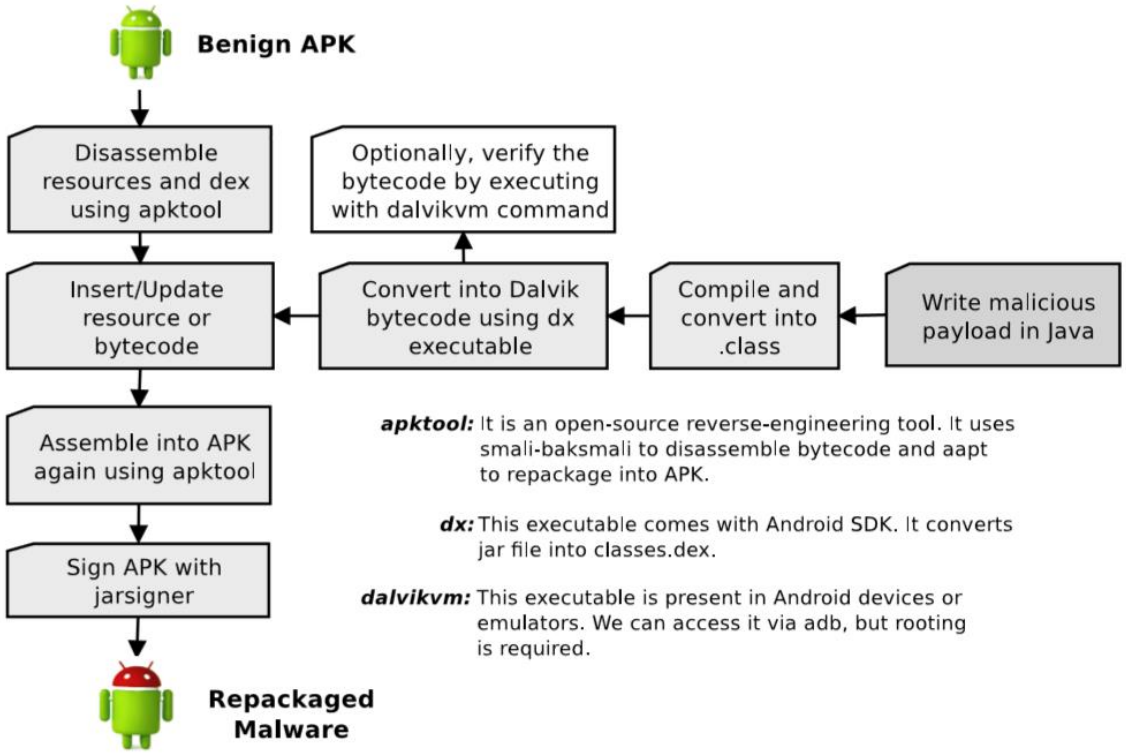

Figura 10. Processo de remontagem do aplicativo [9].

A seção seguinte discute as principais etapas envolvidas na remontagem do aplicativo:

- Baixe o aplicativo de uma loja popular de aplicativos.
- Desmonte o aplicativo com um disassembler, como por exemplo o apktool [54].
- Gere um *payload* malicioso em bytecode Dalvik ou Java, e converta para o bytecode usando a ferramenta Dx [55].
- Adicione o *payload* de *malwares* em um aplicativo benigno. Modifique o AndroidManifest.xml ou outros recursos, se necessário.
- Monte o código fonte modificado usando o apktool.
- Distribua o aplicativo remontado com outro certificado para a loja de aplicativos de terceiros menos monitorada.

A remontagem é uma das técnicas de geração *malware* mais comum. Mais de 80% das amostras do Malware Genome Dataset [56], são variantes de *malwares* remontados da legítima loja de aplicativos oficial. Técnicas de remontagem podem ser usadas para gerar um grande número de variantes de *malware*. Ela também pode ser usada para gerar um número de variantes invisíveis do *malware* já conhecido. Como a assinatura de cada variante de *malware* varia, o *anti-malware* comercial detecta o *malware* invisível. A remontagem é uma grande ameaça, pois ela pode poluir as lojas de distribuição de aplicativos e também prejudica a reputação do desenvolvedor de terceiros. Os autores de *malware* podem desviar a renda das propagandas, substituindo as propagandas dos desenvolvedores originais.

AndroRAT APK Binder [57] remonta e gera uma versão Trojan de um aplicativo popular e legítimo equipando-o com a funcionalidade de acesso remoto. O adversário pode forçar remotamente o dispositivo infectado para enviar mensagens SMS, fazer chamadas de voz, acessar a localização do dispositivo, gravar vídeo ou áudio e acessar os arquivos do dispositivo usando o serviço de acesso remoto.

3.5.2 Envio por download

Um atacante pode empregar a engenharia social, propagandas agressivas e clique em um URL malicioso, para induzir o usuário a baixar *malwares* automaticamente. Opcionalmente, o envio por download pode disfarçar-se de uma aplicação legítima e persuadir o usuário a instalar um aplicativo malicioso. Android/NotCompatible [58] é um notável aplicativo malicioso enviado por download.

3.5.3 Payload dinâmico

Um aplicativo também pode incorporar conteúdo malicioso como um executável (Apk/Jar), em formato criptografado ou simples dentro dos recursos APK. Uma vez

instalado, o aplicativo decifra o *payload*. Se o *payload* é um arquivo Jar, o *malware* carrega a API DexClassLoader e executa o código dinamicamente. No entanto, ele pode persuadir o usuário a instalar o Apk incorporado, disfarçando-se de uma atualização importante. O aplicativo pode executar binários nativos usando a API Runtime.exec, que é equivalente a um fork()/exec() do Linux. As famílias de *malware* BaseBridge [35] e Anserverbot [35] empregam essa técnica. Algumas famílias de *malware* não incorporam conteúdo malicioso como um recurso, mas baixam a partir do servidor remoto e evitam a detecção com êxito. DroidKungFuUpdate [35] é um exemplo notável de execução do *payload* dinamicamente. Essas técnicas passam desapercebidas com métodos de análise estática.

3.6 Malwares: como eles esquivam a segurança?

Nesta seção, serão apresentadas as técnicas de ocultação do estado-da-arte utilizadas pelos aplicativos *malware* para Android. O sistema operacional Android é desenvolvido para um ambiente com restrições de recursos, tendo em conta a disponibilidade limitada de bateria do smartphone subjacente. No dispositivo, aplicativos *anti-malware* não podem executar a análise profunda em tempo real ao contrário de sua versão em desktop. Os autores de *malware* exploram essas restrições de hardware que limitam o *anti-malware* e ocultam os *payloads* nocivos para impedir o *anti-malware* comercial. Técnicas furtivas, como criptografia de código, permutações-chave, *payload* dinâmico, código de reflexão e execução de código nativo continuam a ser um motivo de preocupação para as soluções *anti-malware* baseado em assinatura.

Seguindo as tendências da plataforma desktop, a ocultação de código também está evoluindo no Android [59], [60]. Técnicas de confusão do código são implementadas por uma ou mais das seguintes finalidades.

- Para proteger o algoritmo proprietário de rivais, dificultando a engenharia reversa.
- Para proteger o gerenciamento dos direitos digitais de recursos multimídia para reduzir a pirataria.
- Ofuscando os aplicativos tornando-os compactos e, portanto, mais rápidos na execução.
- Para ocultar o *malware* já conhecido de scanners *anti-malware* a fim de propagar e infectar mais dispositivos.
- Para evitar ou pelo menos atrasar analistas humanos ou motores de análise automática, de descobrir o real motivo do *malware* desconhecido.

O bytecode Dalvik é passível de engenharia reversa, devido à disponibilidade do tipo de informações seguras, tais como tipos de classe, método, definições, variáveis, e também registra Strings e instruções literais. Métodos de transformação de código podem ser facilmente implementados em bytecode Dalvik, mas é possível otimizá-lo com ferramentas de proteção de código, como por exemplo Proguard [61] e Dexguard [62]. Proguard é uma ferramenta de otimização para remover classes, métodos e campos não utilizados. Nomes de classes, métodos, campos, e variáveis locais são substituídos por um código ilegível para dificultar a engenharia reversa. Dexguard é uma ferramenta comercial para proteção do código Android. Ela pode ser usada para implementar técnicas de ofuscação de código, como criptografia de classe, fusão de métodos, criptografia de String, desconfiguração do fluxo de controle, entre outras técnicas, para proteger o aplicativo de ser passivo a engenharia reversa. As técnicas de transformação de código também podem ser utilizadas para impedir as abordagens de detecção de *malware* [59], [60]. [63] propôs um framework de transformação de bytecode Dalvik automatizado para gerar variantes invisíveis de *malware* já conhecidos com diferentes técnicas de ofuscação de bytecode. Além disso, também avaliaram as amostras de *malware* invisíveis contra as melhores técnicas de análise *anti-malware* comerciais e estáticas.

Foi relatado que, mesmo técnicas de transformações triviais podem enganar o *anti-malware* comercial existente.

A seguir, vários métodos de transformação de código utilizados para ofuscar o *malware* conhecido e gerar um grande número de variantes invisíveis do *malware*. A transformação de código também pode ser implementada para impedir as ferramentas de desmontagem [64].

3.6.1 Inserção de código lixo e Reordenação do Opcode

Inserção de código lixo ou código que não realiza nenhuma operação é uma técnica bem conhecida que muda o tamanho do executável e evita de ser reconhecido no banco de dados de assinaturas *anti-malware*. A inserção de código lixo preserva a semântica do aplicativo de origem. No entanto, ele altera a sequência de código de operação (Opcode) para alterar a assinatura do aplicativo *malware*. O código de operação pode ser reordenado com as instruções Goto entre as funções, e alterar o fluxo de controle, preservando a semântica de execução original. Estes métodos podem ser usados para driblar as soluções baseadas em assinatura ou de detecção baseada em código de operação [59], [60].

3.6.2 Renomeação de pacote, classe ou método

O aplicativo Android é exclusivamente identificado com seu nome de pacote único. Por motivo de segurança o bytecode Dalvik preserva os nomes de classes e métodos. Muitos *anti-malware* usam assinaturas triviais, como nome do pacote, classe ou método de um *malware* conhecido como detecção de assinatura [65]. Tais transformações triviais podem ser usadas para driblar o *anti-malware* baseado em assinatura de detecção [60].

3.6.3 Alterando o fluxo de controle

Alguns *anti-malware* usam assinaturas semânticas, como analise de fluxo de controle ou fluxo de dados, para detectar os variantes do *malware* empregando técnicas de transformação simples [65]. O fluxo de controle de um programa pode ser modificado com as instruções Goto, ou inserindo e chamando os métodos lixo. Embora trivial, tais técnicas iludem o *anti-malware* comercial [60].

3.6.4 Criptografia da String

As Strings literais, como mensagens, URLs e comandos de Shell revelam muito sobre o aplicativo. Para evitar esse tipo de análise, as Strings de texto simples podem ser criptografadas se fazendo ilegíveis. Além disso, cada vez que a codificação da String é executada, vários métodos de criptografia dificultam a automatização do processo de decodificação. Nesse caso, Strings literais só podem estar disponíveis durante a execução do código. Por isso, escapam aos métodos de análise estática.

3.6.5 Criptografia da Classe

Informações importantes, tais como checagem de licença de produtos, downloads pagos e DRM podem ser escondidos, criptografando todo as classes que utilizam a informação sensível acima [62].

3.6.6 Criptografia do Recurso

O conteúdo da pasta de recursos, assets e bibliotecas nativas podem ser alteradas tornando-se ilegíveis, portanto, elas devem ser decifradas em tempo de execução [62].

3.6.7 Utilizando APIs de Reflexão

Os métodos de análise estática procuram a API Android dentro dos aplicativos de *malware* para mapear o comportamento malicioso. Os aplicativos de usuário autorizam Java Reflection permitindo a criação de instâncias de classe programáticas ou invocação de método utilizando as Strings literais. Para identificar os nomes exatos das classes ou métodos, pode ser implementada uma análise do fluxo de dados. No entanto, as Strings literais podem ser criptografadas, o que torna difícil a busca automática da API de reflexão. Tais técnicas podem facilmente iludir abordagens de análise estática.

CAPÍTULO 4

MELHORIAS E FERRAMENTAS DE ANÁLISE E DETECÇÃO

Esta seção abrange várias melhorias empregadas pela Android Open Source Project (AOSP) em versões posteriores do Android, como também melhorias desenvolvidas por fabricantes de versões modificadas. A indústria e as universidades têm proposto várias soluções para análise e detecção de *malware* em Android. Nesta seção, também é examinado ferramentas de engenharia reversa e abordagens de detecção promissoras.

4.1 Melhorias de segurança nas versões do Android

Na visão das questões de segurança, as vulnerabilidades ou ataques de *malware* relatados, a AOSP libera patches, atualizações, e melhorias [66]. Aqui, é discutido notáveis correções de segurança e recursos incorporados nas versões posteriores do sistema operacional Android até o Android 5.0 Lollipop:

Android 1.5

- ProPolice para evitar estouros da pilha do buffer (-fstack-protector).
- Safe_iop para reduzir overflows de inteiros.

- Extensões para OpenBSD dlmalloc para evitar vulnerabilidades na função free() e para evitar ataques Chunk Consolidation. Esses ataques são uma maneira comum de explorar a corrupção do heap.
- OpenBSD calloc para evitar overflows de inteiro durante a alocação de memória.

Android 2.3

- Proteções de vulnerabilidades da função Format de Strings (-Wformat-security -Werror=format-security).
- No eXecute (NX) baseado em hardware para impedir a execução de código na pilha e no heap.
- Linux mmap_min_addr para mitigar a dereferência null pointer de escalonamento de privilégios (reforçada no Android 4.1).

Android 4.0

- Address Space Layout Randomization (ASLR) para randomizar posições chaves na memória.

Android 4.1

- Suporte PIE (Position Independent Executable)
- Relocalizações do tipo somente leitura / Binding imediato (-Wl,-z,relro -Wl,-z,now)
- dmesg_restrict habilitado (para evitar vazamento de endereços do *Kernel*)
- kptr_restrict habilitado (para evitar vazamento de endereços do *Kernel*)

Android 4.2

- **Verificação de aplicativos** - Os usuários podem optar por ativar o "Verify Apps" e têm aplicações selecionadas por um verificador de aplicação, antes da instalação. A verificação do aplicativo pode alertar o usuário se eles tentarem instalar um aplicativo que pode ser prejudicial; se um aplicativo é especialmente ruim, ela pode bloquear a instalação.
- **Mais controle de SMS** - Android irá fornecer uma notificação se um aplicativo tentar enviar SMS a um código curto que usa serviços que pode causar custos adicionais. O usuário pode escolher, se deseja permitir que o aplicativo envie a mensagem ou se deseja bloqueá-lo.
- **Sempre no VPN** - VPN pode ser configurado para que os aplicativos não tenham acesso à rede até que uma conexão VPN seja estabelecida. Isso impede que aplicativos enviem dados através de outras redes.
- **Pinagem de Certificado** - As bibliotecas centrais Android agora suportam pinagem de certificado. Domínios pinados recebem uma falha de validação do certificado se o certificado não está vinculado a um conjunto de certificados esperados. Isso protege contra o possível comprometimento de autoridades de certificação.
- **Melhoria da exibição de permissões Android** - As permissões foram organizadas em grupos que são mais facilmente compreendidos pelos usuários. Durante a revisão das permissões, o usuário pode clicar sobre a permissão para ver informações mais detalhadas sobre a permissão.
- **installd mais robusto** - O daemon installd não é executado como o usuário root, reduzindo a superfície de ataque potencial para escalonamento de privilégios de root.

- **script de inicialização mais robusto** - scripts de inicialização, agora, aplicam semântica O_NOFOLLOW para evitar ataques de links simbólicos relacionados.
- **FORTIFY_SOURCE** - Android agora implementa o FORTIFY_SOURCE. Isto é usado por bibliotecas e aplicativos do sistema para evitar a corrupção de memória.
- **Configuração padrão do Content Provider** - As aplicações que têm como alvo a API nível 17 terá a tag "export" definido como "false" por padrão, para cada Content Provider, reduzindo a superfície de ataque padrão para aplicativos.
- **Criptografia** - Modificado as implementações padrão de SecureRandom e Cipher.RSA para usar o OpenSSL. Adicionado suporte SSL Socket para TLSv1.1 e TLSv1.2 usando o OpenSSL 1.0.1.
- **Correções de segurança** – As bibliotecas de código aberto atualizadas com correções de segurança incluem WebKit, libpng, OpenSSL e LibXML. O Android 4.2 também inclui correções para vulnerabilidades especificas do Android. Informações sobre essas vulnerabilidades foram fornecidas para membros da Open Handset Alliance e correções estão disponíveis no Android Open Source Project. Para melhorar a segurança, alguns dispositivos com versões anteriores do Android também podem incluir essas correções.

Android 4.3

- **Sandbox Android reforçado com SELinux**. Este lançamento reforça o Sandbox Android usando o sistema SELinux de controle de acesso obrigatório (Mandatory Access Control - MAC) no *Kernel* do Linux. O reforço SELinux é invisível para os usuários e desenvolvedores, e adiciona robustez ao modelo de segurança do Android existente, mantendo a compatibilidade com as aplicações existentes. Para assegurar a compatibilidade continuada nesta

versão permite o uso de um SELinux no modo permissivo. Este modo registra quaisquer violações de política, mas não vai quebrar aplicações ou afetar o comportamento do sistema.

- **Não há programas setuid/setgid**. Foi adicionado o suporte para recursos de sistema de arquivos para os arquivos do sistema Android e removido todos os programas setuid/setgid. Isso reduz a superfície de ataques à raiz e a probabilidade de potenciais vulnerabilidades de segurança.
- **Autenticação ADB**. Desde o Android 4.2.2, conexões para ADB são autenticadas com um par de chaves RSA. Isso evita o uso não autorizado do ADB, onde o atacante tem acesso físico a um dispositivo.
- **Restringir setuid de aplicativos Android**. A partição do sistema agora está montada em nosuid para processos gerados pelo Zygote, evitando que as aplicações Android executem programas setuid. Isso reduz a superfície de ataque à raiz e a probabilidade de potenciais vulnerabilidades de segurança.
- **Capacidade delimitadora**. O Zygote Android e ADB agora usam prctl(PR_CAPBSET_DROP) para liberar recursos desnecessários antes de executar aplicações. Isso evita que os aplicativos do Android e aplicativos lançados a partir da Shell adquiram capacidades privilegiadas.
- **AndroidKeyStore Provider**. O Android agora tem um provedor de armazenamento de chaves que permite que os aplicativos criem chaves de uso exclusivo. Isso fornece aplicativos com uma API para criar ou armazenar chaves privadas que não podem ser usados por outros aplicativos.
- **KeyChain isBoundKeyAlgorithm**. A API Keychain agora fornece um método (isBoundKeyType) que permite que os aplicativos confirmem que as chaves de todo o sistema são ligadas a uma raiz de confiança para o dispositivo de hardware. Isto proporciona um local para criar ou armazenar as chaves privadas que não podem ser exportadas para fora do dispositivo, mesmo no caso de um comprometimento da raiz.

- **NO_NEW_PRIVS**. O Zygote Android agora usa prctl (PR_SET_NO_NEW_PRIVS) para bloquear a adição de novos privilégios antes da execução do código do aplicativo. Isso evita que os aplicativos do Android realizem operações que podem elevar os privilégios via execve. (Isso requer Linux *Kernel* versão 3.5 ou superior).
- **Melhorias FORTIFY_SOURCE**. Habilitado o FORTIFY_SOURCE em Android x86 e MIPS, e as seguintes chamadas foram fortificadas: strchr(), strrchr(), strlen(), e umask(). Isso pode detectar potenciais vulnerabilidades de corrupção de memória ou constantes String não terminadas.
- **Proteções de relocalização**. Ativado relocalizações do tipo somente leitura (relro) para executáveis estaticamente ligados e foram removidas todas as relocalizações de texto no código do Android. Isso fornece defesa em profundidade contra potenciais vulnerabilidades de corrupção de memória.
- **Melhoria do EntropyMixer**. O EntropyMixer agora escreve a entropia no desligamento/reinicialização do sistema, além de realizar uma mistura periódica. Isso permite a retenção de toda entropia gerada enquanto os dispositivos estão ligados, e é especialmente útil para dispositivos que são reiniciados imediatamente após o provisionamento.

Android 4.4

- **Sandbox Android reforçado com SELinux**. Android agora usa SELinux no modo executável. Isso fornece proteção adicional contra possíveis vulnerabilidades de segurança.
- **VPN por usuário**. Em dispositivos multiusuário, VPNs são agora aplicados por usuário. Isso pode permitir que um usuário encaminhe todo o tráfego de rede através de uma VPN sem afetar outros usuários no dispositivo.

- **ECDSA Provider support in AndroidKeyStore**. O Android agora tem um provedor de armazenamento de chaves que permite o uso de algoritmos ECDSA e DSA.
- **Avisos de monitoramento do dispositivo**. O Android oferece aos usuários um aviso se qualquer certificado foi adicionado ao armazenamento de certificados do dispositivo, que poderia permitir o monitoramento do tráfego da rede criptografada.
- **FORTIFY_SOURCE**. O Android agora suporta FORTIFY_SOURCE nível 2, e todo o código é compilado com essas proteções. FORTIFY_SOURCE foi aprimorado para trabalhar com clang.
- **Pinagem de Certificado**. Android 4.4 detecta e impede o uso de certificados do Google fraudulentos usados em comunicações SSL/TLS seguras.

Android 5.0

- **Criptografado por padrão**. Em dispositivos que acompanham o L out-of-the-box (ou funcionalidades pré-instaladas no dispositivo), a criptografia completa de disco é ativada por padrão para melhorar a proteção de dados em dispositivos perdidos ou roubados. Dispositivos que atualizam para L podem ser criptografados na opção Configurações -> Segurança.
- **Melhoria da criptografia completa de disco**. A senha de usuário é protegida contra ataques de força bruta usando Script e, quando disponível, a chave está ligada ao armazenamento de chaves de hardware para evitar ataques de fora do dispositivo. Como sempre, o segredo da tela de bloqueio do Android e a chave de criptografia do dispositivo não são enviadas para fora do dispositivo ou exposta a qualquer aplicação.
- **Sandbox Android reforçado com SELinux**. O Android agora exige SELinux no modo executável para todos os domínios. Essa nova camada fornece proteção adicional contra possíveis vulnerabilidades de segurança.

- **Smart Lock**. O Android agora inclui trustlets que fornecem mais flexibilidade para desbloquear dispositivos. Por exemplo, trustlets podem permitir que os dispositivos possam ser desbloqueados automaticamente quando próximo a outro dispositivo de confiança (via NFC, Bluetooth) ou sendo usado por alguém com um rosto de confiança (previamente armazenado no dispositivo).
- **Multiusuário, Perfil restrito, e modo de convidado para telefones e tablets**. O Android fornece agora múltiplos usuários em telefones e inclui um modo de convidado, que pode ser usado para fornecer acesso temporário fácil para o seu dispositivo sem a concessão de acesso aos seus dados e aplicativos.
- **Atualizações para WebView sem o OTA (Over The Air – update service)**. O WebView agora pode ser atualizado independente do framework e sem um sistema de OTA. Isto irá permitir uma resposta mais rápida a potenciais problemas de segurança em WebView.
- **Criptografia Atualizada para HTTPS e TLS/SSL.** TLSv1.2 e TLSv1.1 agora está ativada, o modelo Forward Secrecy foi adotado, AES-GCM agora está ativado, e os conjuntos de cifras (MD5, 3DES, e conjuntos de cifras exportados) são agora desativados.
- **non-PIE linker support removido**. O Android agora exige que todos os executáveis ligados dinamicamente suportem PIE (position-independent executables). Isso melhora a implementação do ASLR do Android.
- **Melhorias no FORTIFY_SOURCE**. As seguintes funções libc agora implementam proteções FORTIFY_SOURCE: strcpy(), strncpy(), read(), recvfrom(), FD_CLR(), FD_SET() e FD_ISSET(). Isso fornece proteção contra vulnerabilidades de corrupção de memória envolvendo essas funções.

4.2 Melhorias de segurança por terceiros

Muitos aperfeiçoamentos de segurança Android independentes têm sido propostos [67]-[70]. Esses mecanismos permitem uma organização para criar políticas de segurança de granularidade fina para os dispositivos de seus funcionários. A informação contextual, como a localização do dispositivo, as permissões de aplicativos e de comunicação entre os aplicativos pode ser monitorada e verificada contra as políticas já declaradas.

4.3 Ferramentas de engenharia reversa

O conteúdo do pacote Android (APK) é armazenado no formato binário. Antes de iniciar a tarefa de avaliação, análise ou detecção, é importante desmontá-lo para efetuar um processamento adicional. Há uma série de ferramentas para desmontar ou decompilar o aplicativo Android. Na seção seguinte, são discutidas algumas ferramentas de engenharia reversa conhecidas considerando seus pontos fortes.

- apktool [54] pode decodificar o conteúdo binário de um APK em forma quase original na estrutura de diretório do projeto. Ele desmonta os recursos binários e converte o bytecode dentro de classes.dex no bytecode smali [71] para facilitar a leitura e manipulação. Após fazer as alterações, ele também pode ré empacotá-lo de volta em um APK. Esta ferramenta é uma das melhores ferramentas de engenharia reversa de código aberto.

- dex2jar [72] é um disassembler para analisar tanto o .dex como o arquivo dex otimizado, fornecendo uma API leve para acessá-lo. dex2jar também pode converter o arquivo dex para um arquivo jar, por re-segmentação o Dalvik bytecode em Java bytecode, para posterior manipulação. Além disso, ele também pode remontar o jar em uma .dex após as modificações.

- O projeto Dare [73] visa re-segmentar o Dalvik bytecode dentro de classes.dex para os arquivos .class tradicionais, usando um forte algoritmo de inferência de tipos. Estes arquivos .class podem ser ainda analisados utilizando uma variedade de técnicas tradicionais desenvolvidas para aplicações Java, incluindo os de-compiladores. [74] demonstra que Dare é 40% mais preciso do que dex2jar.

- Dedexer [75] desmonta o classes.dex em sintaxe do tipo Jasmin e cria um arquivo separado para cada classe, mantendo a estrutura do diretório do pacote para facilitar a leitura e manipulação. No entanto, ao contrário do apktool, não pode remontar os arquivos class intermediários desmontados.

- JEB [76] é o principal software profissional de engenharia reversa Android disponível em plataformas Windows, Linux e Macintosh. É um decompilador interativo baseado em GUI para analisar o conteúdo revertido de aplicativos *malwares*. Informações do aplicativo tal como o manifesto, recursos, certificados, Strings literais, podem ser examinados em conteúdo Java, proporcionando uma navegação fácil através das referências cruzadas. JEB converte o Dalvik bytecode diretamente na fonte Java, utilizando a semântica do Dalvik bytecode. Excepcionalmente, JEB também pode de-ofuscar o Dalvik bytecode para tornar o código desmontado mais legível em comparação com os seus homólogos [54], [72]. JEB suporta scripts em Python ou plugins, permitindo o acesso ao código Java decompilado Abstract Syntax Tree (AST), através da API. Este recurso é útil para automatizar a análise personalizada.

4.4 Androguard

Androguard [65] é uma ferramenta open-source de análise estática, que é capaz de fazer engenharia reversa de desmontar e decompilar aplicativos Android. O objetivo desta ferramenta é avaliação de risco, análise e detecção de *malwares*. Ela gera os gráficos do fluxo de controle para cada método e fornece acesso na linha de comando e interface gráfica através da API Python. A abordagem Normalized Compression Distance (NCD) do Androguard busca semelhanças e diferenças dos dois clones suspeitos de maneira confiável, o que também é útil para detectar aplicativos ré empacotados. Ele fornece APIs python para acessar os recursos desmontados e as estruturas de análise estática como blocos básicos, fluxo de controle e instruções de um APK. Um analista pode desenvolver sua própria estrutura de análise estática usando as APIs python. A seguir estão algumas das características explicadas abaixo.

- **Similaridade do código do aplicativo**: Androguard busca semelhanças entre dois aplicativos calculando o NCD entre cada pares de métodos e calcula uma pontuação de semelhança entre 0-100, em que 100 significa aplicativos idênticos. Ele exibe os métodos IDENTICAL, SIMILAR, NEW, DELETED e SKIPPED dos dois clones suspeitos. Da mesma forma, ele exibe as diferenças entre os dois métodos, comparando cada pares de blocos básicos. Mais especificamente, para calcular as diferenças entre os dois métodos similares, primeiro converte cada instrução única no bloco básico em uma String. Em seguida, aplica-se o algoritmo Longest Common Subsequence nestas cadeias de dois blocos básicos para buscar diferenças entre eles [77].

- **Indicador de Risco**: O indicador de risco calcula a pontuação de risco de um APK de 0 (baixo risco) a 100 (de alto risco). Ele considera os parâmetros a seguir:

- Nativo, Reflexão, presença de código criptográfico e dinâmico em um aplicativo.
- Número de arquivos executáveis, bibliotecas compartilhadas presentes em um aplicativo.
- Pedidos de permissão relacionados com a privacidade e os riscos monetários.
- Outras solicitações de permissão (Perigosas, Sistema, Assinatura).

- **Assinatura de Aplicativos maliciosos**: O Androguard gerencia um banco de dados de assinaturas e fornece uma interface para adicionar ou remover assinaturas de ou para o banco de dados. A assinatura é descrita no formato JSON. Ela contém um nome (ou nome de família), conjunto de sub-assinaturas e uma fórmula booleana para misturar diferentes sub-assinaturas. Seguem-se os dois tipos de sub-assinaturas:
  - METHSIM: Ele contém três parâmetros, CN - nome da classe, MN - nome do método e D - descritor.
  - CLASSSIM: Ele contém um parâmetro único, CN - nome da classe.

  Assim a sub-assinatura pode ser aplicada em um método específico ou na classe inteira. Diferentes sub-assinaturas podem ser misturadas com fórmula booleana (BF).

4.5 Bouncer

A Google protege a sua própria loja de aplicativos, Google Play, com um sistema chamado de Bouncer. É uma plataforma de análise dinâmica baseada em máquina virtual para testar os aplicativos enviados de desenvolvedores de terceiros, antes de tornarem disponíveis aos usuários para download. Ela executa o aplicativo para procurar qualquer comportamento malicioso e também compara com aplicativos

maliciosos previamente analisados. Embora nenhuma documentação de funcionamento interno está disponível, [19] apresenta a análise do ambiente Bouncer através da implementação de um aplicativo de comando e controle personalizado. Técnicas de código de *payload* dinâmico podem iludir o exame minucioso do Bouncer [78].

4.6 DroidMoss

O DroidMOSS [56] é um protótipo de detecção de aplicativos ré empacotados, empregando medidas de similaridade de semântica dos arquivos. Mais especificamente, ele extrai a sequência do código de operação DEX de um aplicativo e gera uma assinatura difusa refazendo [79] a assinatura do código de operação. Ele também adiciona informações do certificado do desenvolvedor, mapeados na assinatura em um identificador único de 32 bits. Características de aplicativos suspeitos são verificadas contra os aplicativos originais usando o algoritmo edit-distance para identificar a pontuação de similaridade. A intuição por trás do DroidMOSS usando o mecanismo dos códigos de operação é, pode ser fácil para os adversários modificar os operandos, mas muito difícil mudar os códigos de operações reais [56]. Esta abordagem tem várias desvantagens. Em primeiro lugar, ela só considera bytecode DEX, ignorando o código nativo e os recursos do aplicativo. Em segundo lugar, a sequência de código de operação não consiste em informação semântica de alto nível e, consequentemente, gera falsos negativos. O adversário inteligente pode facilmente iludir esta técnica utilizando técnicas de transformação de código como a inserção de bytecode inútil, reestruturar métodos e alterar o controle de fluxo para evadir o protótipo DroidMOSS.

CAPÍTULO 5

DEFESAS

Esta seção discute as defesas que podem ser tomadas no Android com base no estudo realizado neste trabalho. São abordadas questões tanto do ponto de vista do programador, a fim de desenvolver um aplicativo mais seguro, como do usuário, com o objetivo de se proteger contra ameaças.

5.1 Como desenvolver um aplicativo mais seguro?

A fim de desenvolver um aplicativo mais seguro o programador deve conhecer bem a arquitetura do Android (Capítulo 2), as ameaças que o cercam e como elas se infiltram no sistema (Capítulo 3), as melhorias e técnicas de análise e detecção (Capítulo 4), para saber como se prevenir e defender-se.

Como citado no Capítulo 4, existem diversas técnicas para análise e detecção de *malwares*, e também ferramentas para avaliar o risco ao qual um aplicativo está sujeito. A Google, recomenda fortemente que o desenvolvedor teste o seu aplicativo em uma dessas ferramentas, como o Androguard, que avalia o aplicativo e fornece uma pontuação de risco. Há muitas outras ferramentas conhecidas do estado-da-arte para avaliação do aplicativo.

O desenvolvedor deve estar sempre atualizado as melhores práticas de programação para Android segundo a AOSP, e buscar sempre implementar um aplicativo tendo como alvo a API de nível mais alta (ou mais nova) do Android. Desta

maneira, o sistema vai usufruir das técnicas de segurança mais novas implementadas pela AOSP, conforme citado na subseção 4.1.

É de extrema importância que o programador tenha conhecimento das vulnerabilidades já conhecidas do Android, em [30] é exibida uma lista detalhando os problemas encontrados das versões posteriores do Android até a mais recente. Dado que o programador conhece as vulnerabilidades, é possível desenvolver um aplicativo mais seguro esquivando-se de tais problemas já conhecidos.

Para os programadores de versões modificadas do Android, a bluebox realizou uma análise com diversos dispositivos Android de fabricantes diferentes, como citado na subseção 3.2. Os resultados desta análise apontam problemas comuns que os fabricantes originam ao modificar o sistema operacional, estes resultam em vulnerabilidades no sistema. É uma boa prática, observá-los minuciosamente a fim de evitar cometer os mesmos erros.

O Android fornece uma API com recursos de criptografia. É recomendado desenvolver a aplicação utilizando destes recursos para criptografar os dados do usuário. Portanto, ainda que um *malware* consiga se infiltrar no sistema e obter os dados do usuário, haverá uma camada adicional de segurança, pois os dados estarão criptografados, e a menos que o adversário conheça a chave, estes dados estarão protegidos.

A AOSP recomenda que o desenvolvedor minimize o número de permissões que seus aplicativos pedem. Não ter acesso a permissões sensíveis reduz o risco de, inadvertidamente, ocorrer um mau uso de tais permissões. Faltando essas permissões, pode melhorar a adoção do usuário, e faz seu aplicativo menos vulnerável a ataques. Se a permissão não é necessária para o aplicativo, o desenvolvedor não deve solicitá-la.

É possível distribuir um aplicativo Android em mercados de terceiros ou através de fontes desconhecidas (diretamente no cabo USB). Contudo, o aplicativo fica

completamente vulnerável a ameaças, podendo ser remontado e nele inserido um código malicioso, como citado na subseção 3.5.1. Por esta razão, a Google recomenda distribuir o aplicativo na sua loja oficial Google Play, que por sua vez é examinado pelo Bouncer, seguindo as etapas de preparação e publicação do aplicativo, conforme descrito na subseção 2.7. Contudo, mesmo que o desenvolvedor não tenha a intenção de publicar o seu aplicativo na loja oficial, é recomendado que seja gerado um certificado comercial para distribuição do aplicativo, caso contrário, o aplicativo será distribuído com uma chave de teste, que está sujeita a privilégios de usuário root.

5.2 Como o usuário pode se proteger?

Esta seção tem o propósito de apresentar medidas de segurança e prevenção ao usuário de aplicativos Android, tendo em vista que existem diversas ameaças no universo Android, conforme mencionado no Capítulo 3.

O primeiro grande passo é estar sempre com a versão mais atualizada possível do Android. Por meio de intervalos curtos de tempo, cerca de meses, a Google desenvolve um patch novo para o seu sistema operacional móvel. Estes patchs tem o objetivo de trazer várias melhorias no Android, inclusive melhorias nas questões de segurança. Diversas vulnerabilidades foram resolvidas a cada nova versão do Android, conforme citado na subseção 4.1. Uma das principais razões pelas quais o usuário se torna alvo de ataques no Android, é por usar versões antigas do sistema operacional. Outro grande problema, é a fragmentação, como foi descrito na subseção 3.2. Para usuários de versões modificadas pelo fabricante, recomenda-se estar sempre checando por atualizações do sistema operacional.

A maneira mais comum de adquirir *malwares*, é através da instalação de aplicativos por meio de fontes desconhecidas ou mercados de terceiros, pois estes não garantem segurança alguma para o usuário. A Google recomenda que os

usuários sempre instalem aplicativos a partir da sua loja oficial Google Play, a qual realiza uma análise dinâmica de seus aplicativos por meio da ferramenta Bouncer. No entanto, isto não vai garantir que o aplicativo seja 100% seguro, pois já houve casos em que *malwares* se infiltraram na loja oficial, mas certamente oferecerá uma camada adicional de segurança em relação a outras formas de instalação.

O sistema baseado em permissões pela Android, é uma maneira de fornecer ao usuário certo controle sobre o que o aplicativo pode realizar em seu dispositivo. Aplicativos não podem realizar ações que não sejam autorizadas pelo usuário no momento da instalação. Portanto, o usuário tem a responsabilidade de observar atentamente as permissões requisitadas pelo aplicativo no momento da instalação, e caso haja permissões perigosas ou permissões que pareçam desnecessárias a aplicação, decidir se deve instalar ou não o aplicativo.

No caso de aplicativos que mechem com dados sensíveis, como OnlineBanking, senhas de cartão de credito ou dados confidenciais, é fortemente recomendado que o usuário procure a empresa ou instituição responsável pelo aplicativo, com o objetivo de confirmar se o aplicativo realmente é da empresa. Pois, como citado na subseção 3.4.1, existem aplicativos maliciosos que se disfarçam de aplicativos comuns, conhecidos como Trojans, para roubar os dados do usuário.

Com o propósito de melhorar a segurança, a Google fornece um aplicativo chamado Verify Apps. Ele tem o objetivo de realizar uma verificação nos aplicativos instalados por fontes desconhecidas, prevenindo o sistema de *malwares*. Também é uma boa prática instalar aplicativos para monitoramento e defesa do dispositivo, como um antivírus. Apesar de não serem tão eficazes pela limitação que a plataforma Android impõe (Capítulo 2), eles fornecem uma proteção extra ao usuário.

A versão mais recente do Android, ou Android Lollipop 5.0, disponibiliza ao usuário uma nova característica de multiusuários, descrito na subseção 4.1. Com

isso, por exemplo, o usuário pode ter um perfil para a empresa e um perfil de uso pessoal. Os dados de cada perfil são protegidos por um Sandbox diferente, fazendo com que o uso pessoal do usuário não coloque em risco os dados da empresa.

Mais alternativas para prevenção e proteção do usuário Android são apresentadas pela bluebox, que escreveu um guia de segurança para o usuário Android [80]. Embora sejam seguidas estas orientações, o usuário não garante a sua segurança, mas diminui em muito o risco a ameaças em seu dispositivo.

CAPÍTULO 6

APLICATIVO DESENVOLVIDO E TESTES

Esta seção, tem o propósito de apresentar o aplicativo desenvolvido para este trabalho, com o fim de avaliar os problemas citados na segurança do Android e mostrar os resultados obtidos.

6.1 Aplicativo Desenvolvido

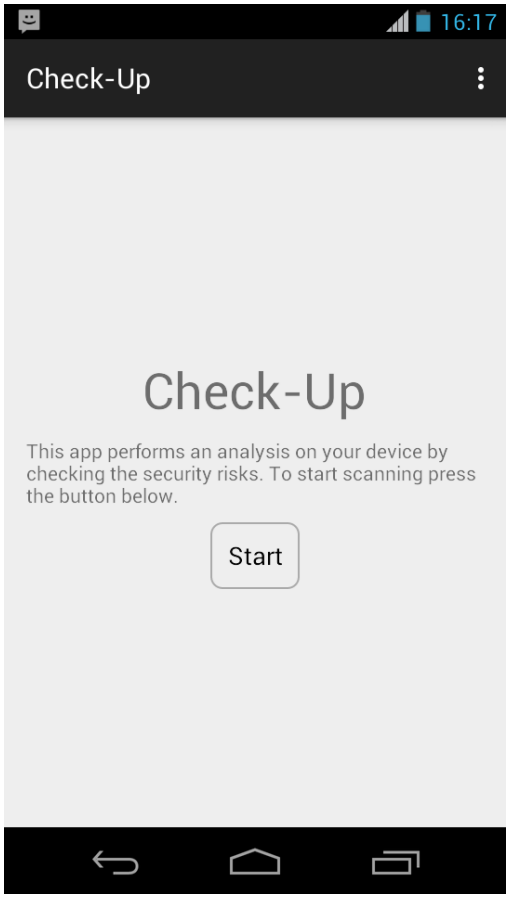

Figura 11. Check-Up: Tela Inicial.

Neste trabalho, foi desenvolvido o aplicativo Check-Up (Figura 11). Uma vez instalado, é responsável por realizar uma análise no dispositivo do usuário, com o objetivo de verificar questões de segurança, como:

- Vulnerabilidades da arquitetura do sistema operacional, com base em [30].
- Configurações de segurança
    - Opções do programador
    - Depuração USB
    - Instalação de aplicativos de fontes desconhecidas
    - Criptografia do dispositivo
- Se o Google Playstore está instalado no dispositivo.

Após a execução da análise (Figura 12), através do botão “Start”, o aplicativo exibe os resultados para o usuário (Figura 13).

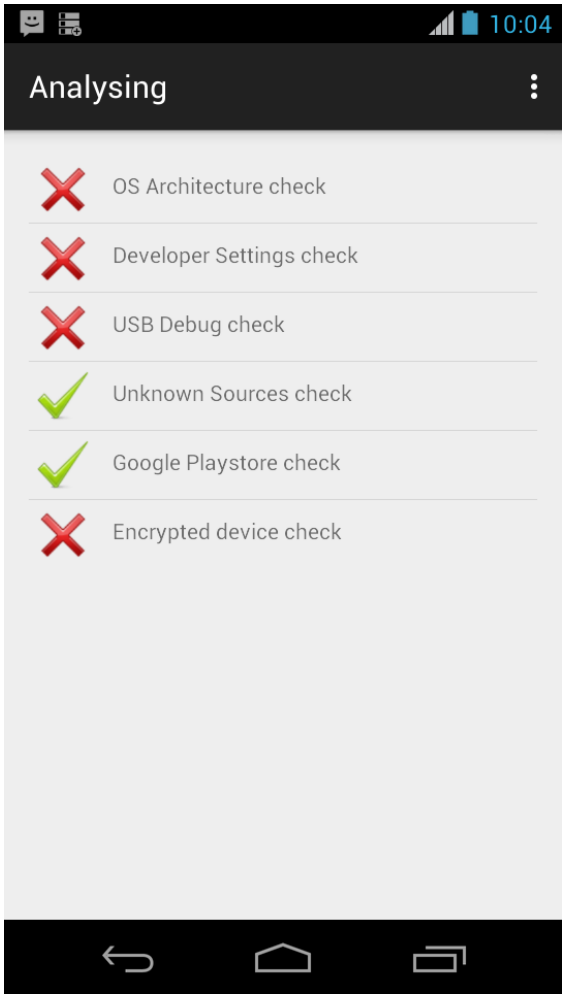

Figura 12. Check-Up: Tela de Análise.

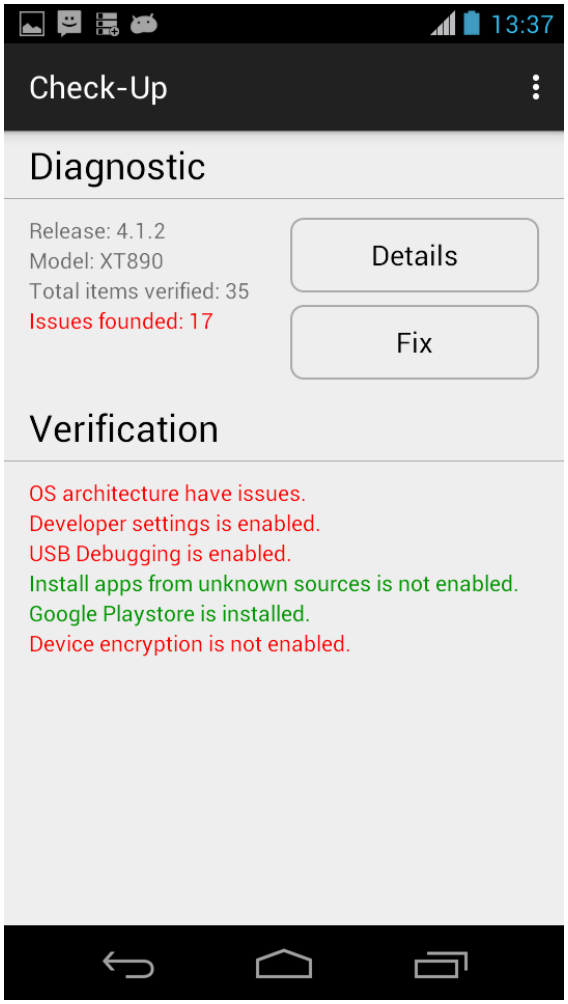

Figura 13. Check-Up: Tela de relatório da análise.

É exibido informações características do dispositivo, como versão do Android instalado e o modelo. Além disso, os itens que foram verificados e quantos problemas foram encontrados. O resultado da verificação apresenta em verde os itens que passaram no teste, e em vermelho o que precisa ser corrigido. Adicionalmente, é possível verificar os detalhes dos problemas encontrados por meio do botão “Detail” (Figura 14).

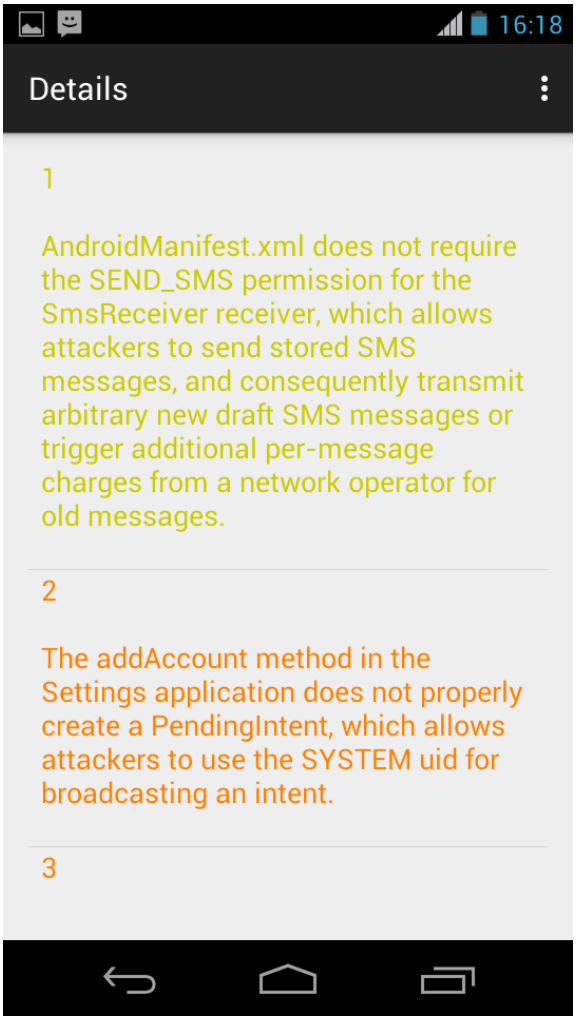

Figura 14. Check-Up: Tela de detalhes do relatório.

Para cada problema é atribuído uma nota de 0 a 10 de acordo com a sua gravidade. Com base nessa nota, é atribuído uma cor ao texto de cada problema.

- De 0 a 3: Texto amarelo.
- De 4 a 7: Texto laranja.
- De 8 a 10: Texto vermelho.

Há também a opção de corrigir os problemas, através do botão “Fix” (Figura 15), que por sua vez fornecerá ao usuário um passo a passo de como fixar os problemas encontrados.

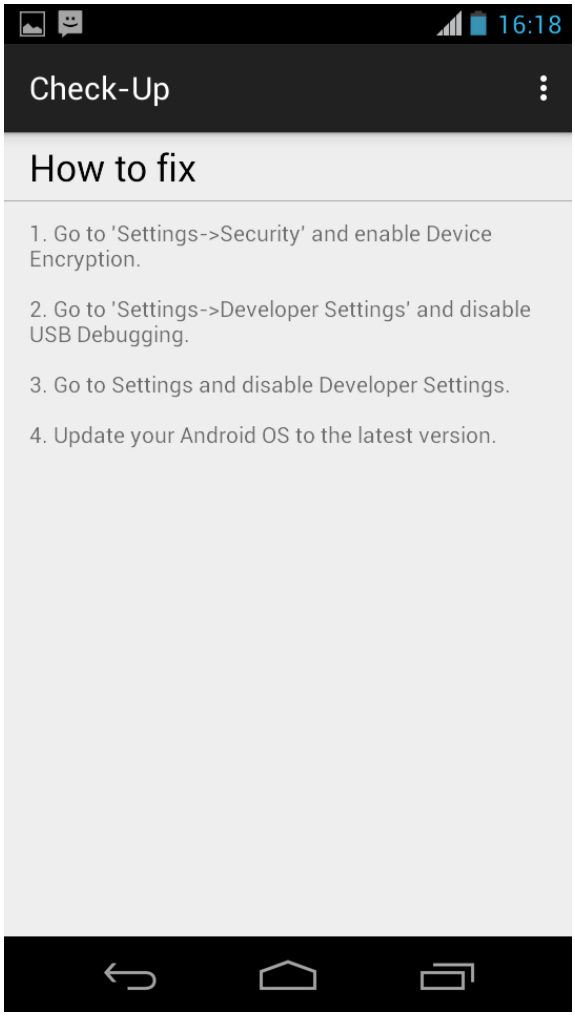

Figura 15. Check-Up: Tela de correção dos problemas.

6.2 Testes

Com o propósito de avaliar o aplicativo desenvolvido foram realizados vários testes em diversos dispositivos, com diferentes versões do Android instaladas, e os seus resultados estão registrados nesta seção. Através do relatório de análise do Check-Up, é possível observar que quase todos os dispositivos testados falharam no quesito vulnerabilidades do sistema operacional, com exceção do Teste 6, que possui a versão mais nova do Android instalada, que por sua vez, ainda não há vulnerabilidades conhecidas. Todos os testes apresentaram problemas nas

configurações de segurança, que podem ser fixados seguindo o guia de correção do aplicativo.

Teste 1

Dispositivo: Motorola Razr i (XT890)

Android: 4.1.2

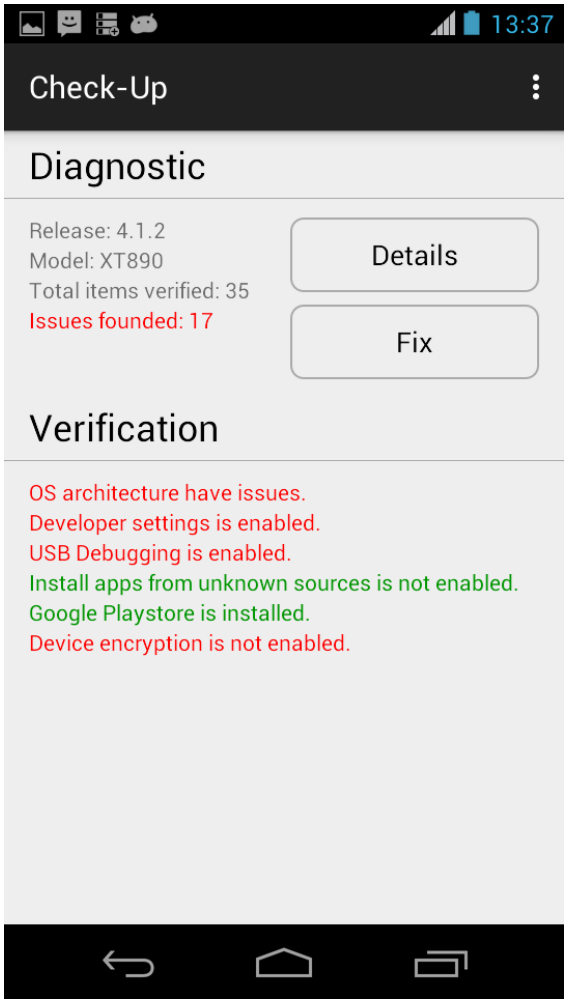

Figura 16. Teste 1.

Teste 2

Dispositivo: Samsung Galaxy Note 2 (GT-N7100)

Android: 4.4.2

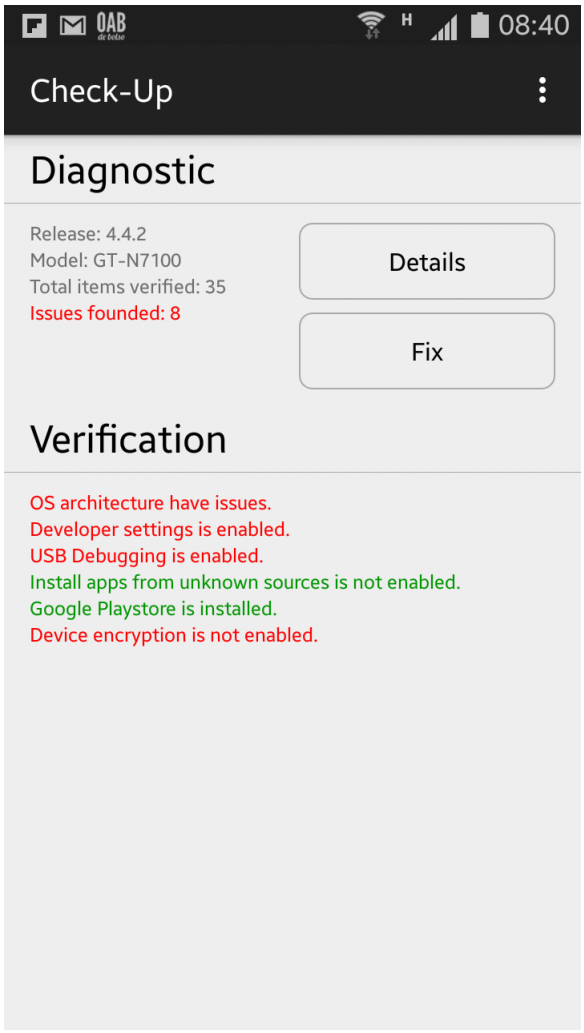

Figura 17. Teste 2.

Teste 3

Dispositivo: Samsung Galaxy Young Duos (GT-S530B)

Android: 4.0.4

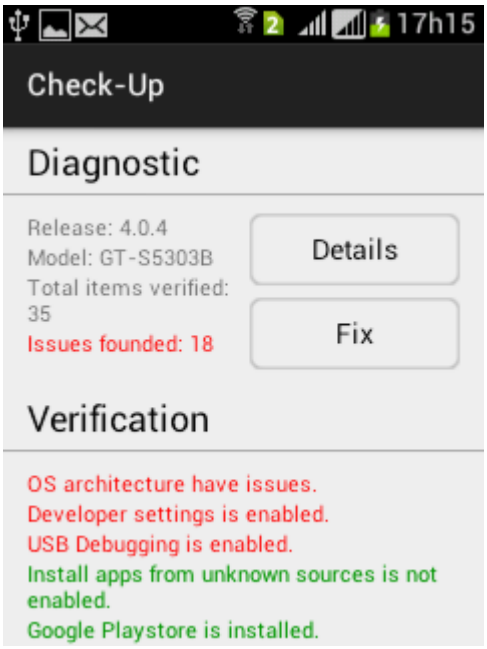

Figura 18. Teste 3.

Teste 4

Dispositivo: Samsung Galaxy S3 Neo (GT-I9300I)

Android: 4.4.4

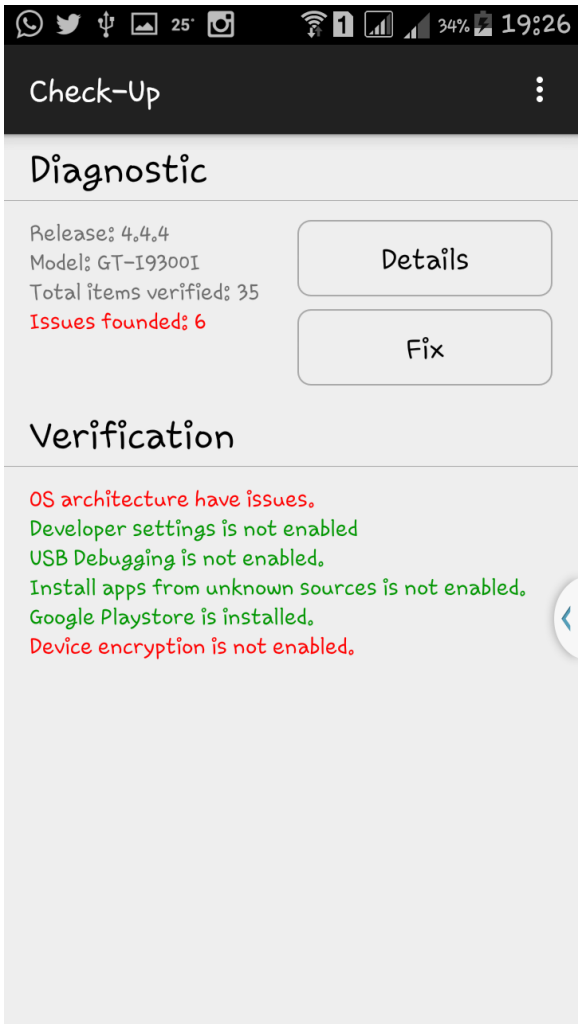

Figura 19. Teste 4.

Teste 5

Dispositivo: Moto G – 2 Geração (XT1078)

Android: 5.0.2

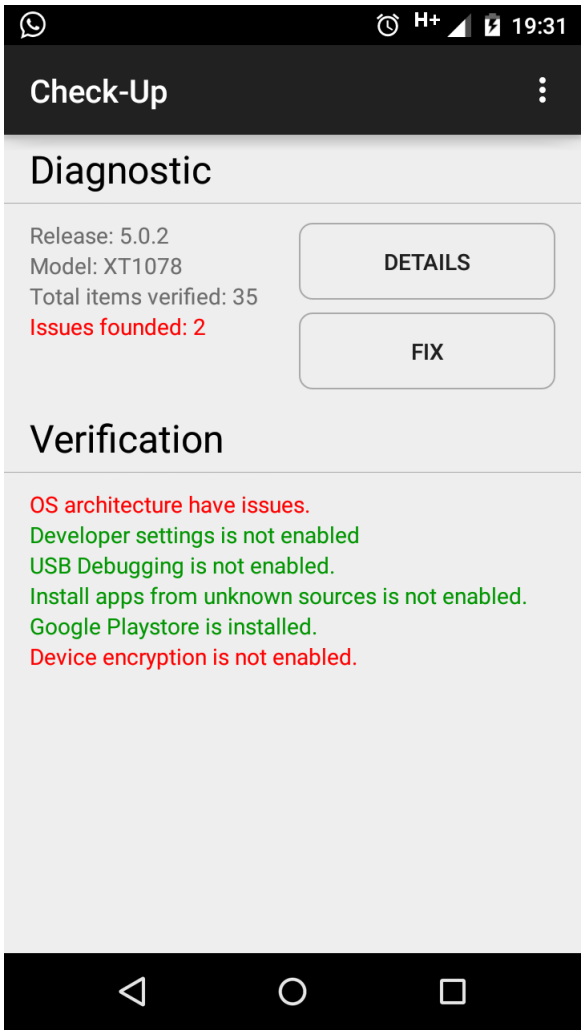

Figura 20. Teste 5.

Teste 6

Dispositivo: Moto X (XT1097)

Android: 5.1

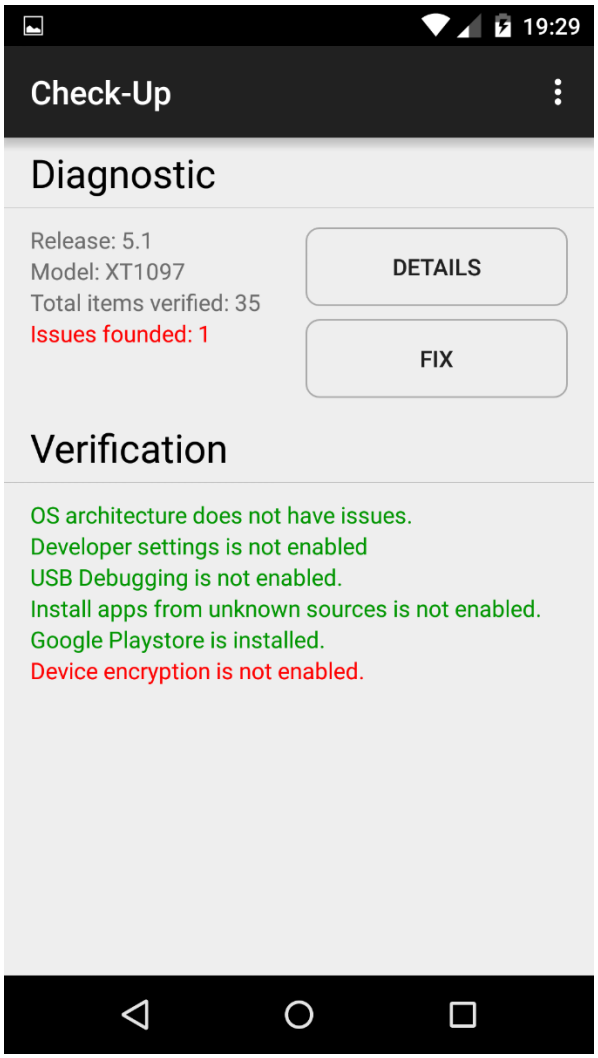

Figura 21. Teste 6.

Teste 7

Dispositivo: Samsung Galaxy S4 (GT-I9515L)

Android: 4.4.2

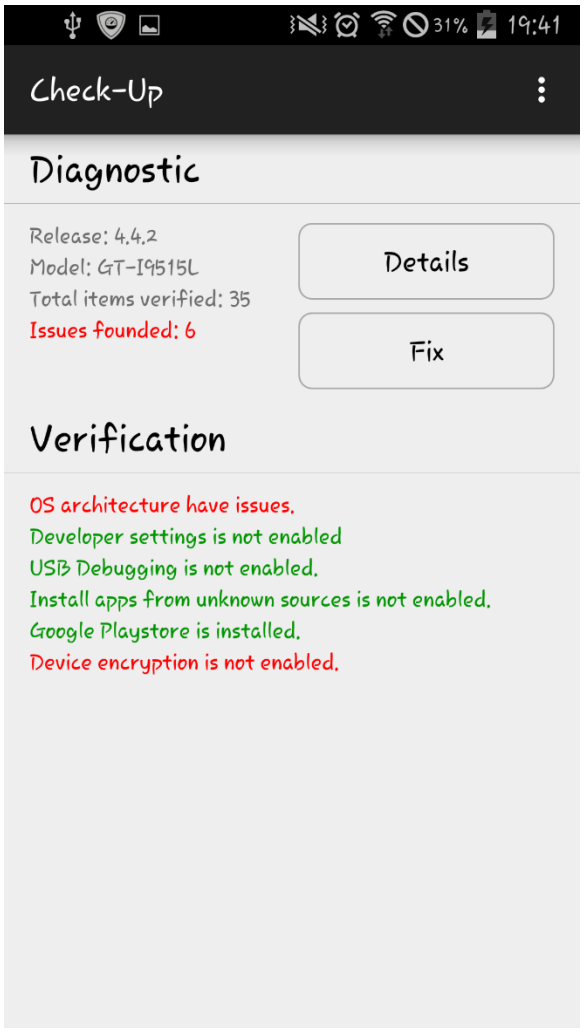

Figura 22. Teste 7.

Teste 8

Dispositivo: Samsung Galaxy S3 (GT-I9300I)

Android: 4.4.4

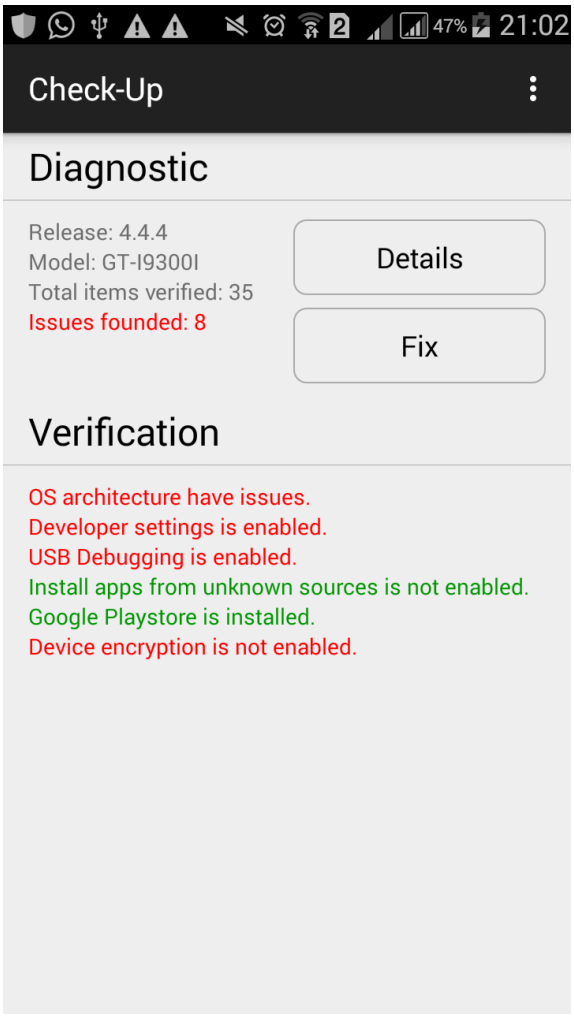

Figura 23. Teste 8.

Teste 9

Dispositivo: Moto G – 1 Geração (XT1033)

Android: 4.4.4

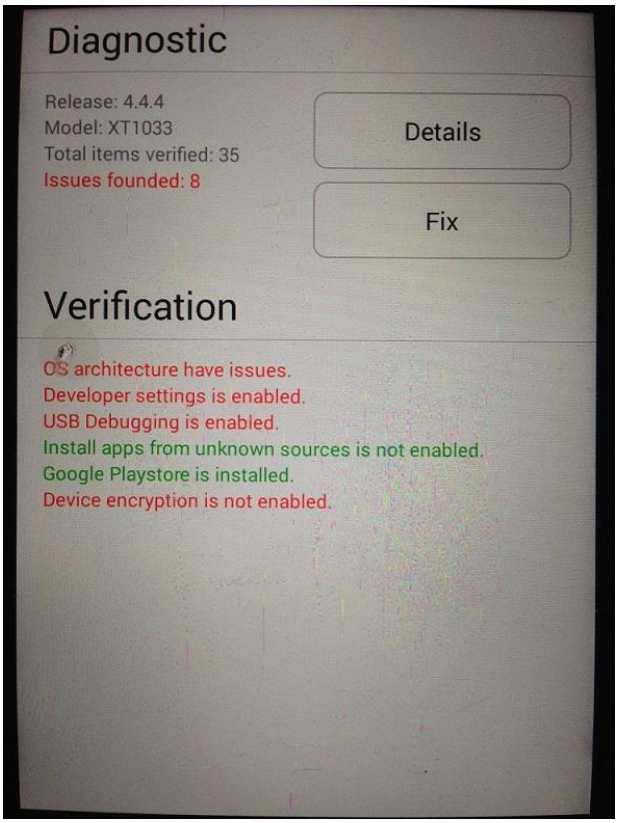

Figura 24. Teste 9.

Teste 10

Dispositivo: Sony XPERIA T3 (D5106)

Android: 4.4.2

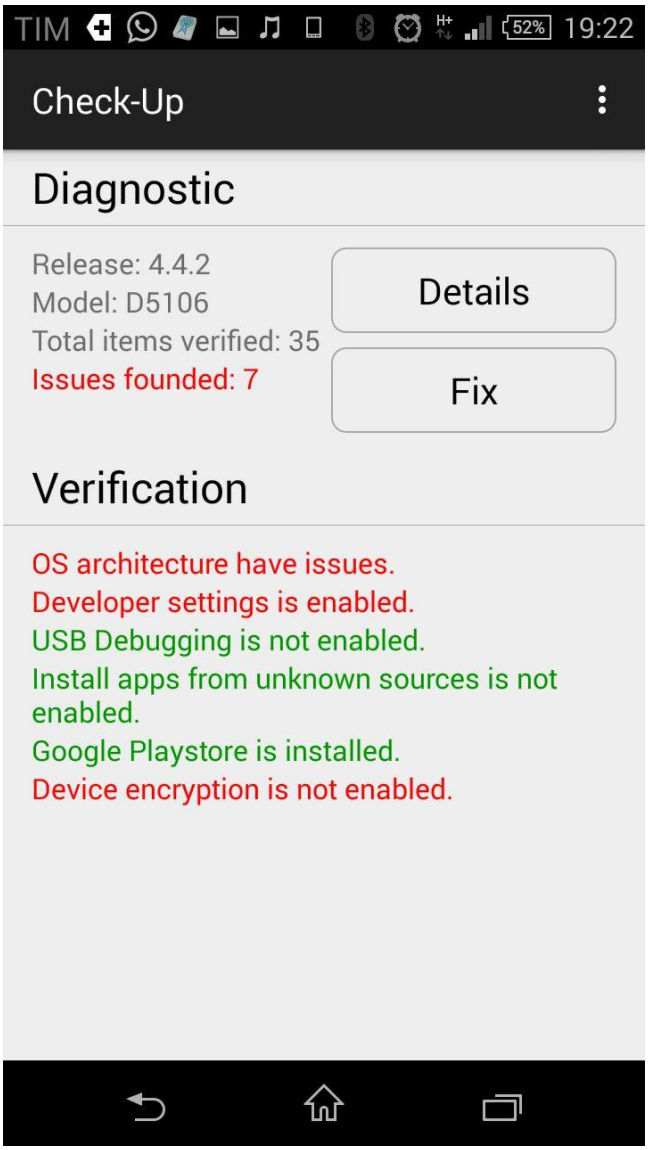

Figura 25. Teste 10.

CONCLUSÕES

A plataforma Android é alvo de mais de 90% dos ataques de *malwares* a smartphones e tablets nos últimos anos. E isto acontece primeiramente por ser o sistema operacional móvel mais utilizado do mundo. Em segundo lugar, por ser um projeto de código aberto, de forma que os adversários podem estudar as vulnerabilidades do sistema. Em consequência de o código estar aberto e livre para modificações por terceiros, a fragmentação do sistema têm causado um grande impacto na segurança do Android. Como resultado, por causa da incompatibilidade das versões, a plataforma atualmente não consegue desenvolver uma solução única para resolver questões de segurança.

O universo Android está sujeito aos mais variados tipos de ameaças, que por sua vez, exploram as suas vulnerabilidades. Existem aplicativos que se comportam aparentemente de maneira inocente, como Trojans, mas que realizam comportamentos maliciosos sem o consentimento do usuário. Há também aqueles que invadem o ambiente pessoal do usuário de forma intrusiva com propagandas, como Agressive Adware. Além disso, existem outros *malwares* conhecidos aos quais o Android está sujeito, como mencionado no Capítulo 3. Neste trabalho, foram apresentadas estas ameaças, suas técnicas de infiltração, e como elas se ocultam no sistema.

Desde a primeira versão do Android até a mais recente, diversas melhorias de segurança foram implementadas, visando eliminar as vulnerabilidades conhecidas e garantir ao usuário um ambiente onde seus dados podem estar protegidos. Todavia,

ainda há brechas que continuam sendo exploradas pelos adversários. Muitas ameaças cercam o Android, felizmente, há maneiras de construir uma aplicação mais segura. Para isso, há uma breve descrição de boas práticas a serem seguidos pelo programador no Capítulo 5. Além disso, ainda no Capítulo 5, o usuário pode utilizar de vários passos para se prevenir contra ameaças e se proteger.

Com o propósito de fornecer mais segurança ao usuário, foi desenvolvido neste trabalho o aplicativo Check-Up. Este app é responsável por analisar as questões de segurança mencionadas ao longo deste trabalho. Essa análise é apresentada ao usuário juntamente com um tutorial de como resolver cada problema encontrado, tornando o dispositivo mais seguro. Foram realizados vários testes em diferentes versões do Android, e em todos eles o aplicativo foi capaz de detectar as vulnerabilidades no dispositivo para que sejam corrigidas pelo usuário segundo o passo a passo fornecido pelo próprio aplicativo.

REFERÊNCIAS

[1] **Smartphone OS Market Share**. Disponível em:

<http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. Acesso em 15 de abril. 2015.

[2] **O problema crescente do malware no Android**. Disponível em:

<http://blog.kaspersky.com.br/o-problema-crescente-do-malware-no-android/3044/>. Acesso em 15 de abril. 2015.

[3] **Android security remains a glaring problem 10 reasons why**. Disponível em:

<http://www.eweek.com/mobile/slideshows/android-security-remains-a-glaring-problem-10-reasons-why.html>. Acesso em 15 de abril. 2015.

[4] **A clear eyed guide to Androids actual security risks**. Disponível em:

<http://www.infoworld.com/article/2609338/android/a-clear-eyed-guide-to-android-s-actual-security-risks.html>. Acesso em 15 de abril. 2015.

[5] BING HAN. **Analysis and Research of System Security Based on Android**. China, 2012.

[6] **Android Architecture**. Disponível em:

<http://www.tutorialspoint.com/android/android_architecture.htm>. Acesso em 15 de abril. 2015.

[7] Google Inc. **System Architecture**. Disponível em:

<http://developer.android.com/images/system-architecture.jpg>. Acesso em 15 de abril. 2015.

[8] Google Inc. **Android Security Overview**. Disponível em:

<http://source.android.com/devices/tech/security>. Acesso em 15 de abril. 2015.

[9] FARUKI P.; BHARMAL A.; LAXMI V.; GANMOOR V.; GAUR M.; CONTI M.; RAJARAJAN M. . **Android Security: A Survey of Issues, Malware Penetration and Defenses**. IEEE, 2015.

[10] Google Inc. **Android Fundamentals**. Disponível em:

<http://developer.android.com/guide/components/fundamentals.html>. Acesso em 15 de abril. 2015.

[11] **Android Application Components**. Disponível em:

<http://www.tutorialspoint.com/android/android_application_components.htm>. Acesso em 15 de abril. 2015.

[12] ENCK W.; ONGTANG M.; MCDANIEL P. . **Understanding Android security**. IEEE, 2009.

[13] Google Inc. **Android Development - Intent**. Disponível em:

<http://developer.android.com/reference/android/content/Intent.html>. Acesso em 15 de abril. 2015.

[14] BERGER B. J.; BUNKE M.; SOHR K. . **An Android Security Case Study with Bauhaus**. Germany, 2011.

[15] **Android Kernel Features**. Disponível em:

<http://elinux.org/Android Kernel Features>. Acesso em 15 de abril. 2015.

[16] Google Inc. **Permission-element**. Disponível em:

<http://developer.android.com/guide/topics/manifest/permission-element.html>. Acesso em 15 de abril. 2015.

[17] SANZ B.; SANTOS I.; LAORDEN C.; UGARTE X.; BRINGAS P. G.; ALVAREZ G. . **PUMA: Permission Usage to detect Malware in Android**. Springer, 2013.

[18] **Google Bouncer: Protecting the google play market**. Disponível em:

<http://blog.trendmicro.com/trendlabs-security-intelligence/a-lookat-google-bouncer/>. Acesso em 15 de abril. 2015.

[19] OBERHIDE JON. **Dissecting the Android Bouncer**. Disponível em:

<http: //jon.oberheide.org/blog/2012/06/21/dissecting-the-android-bouncer/>. Acesso em 15 de abril. 2015.

[20] **Exercising Our Remote Application Removal Feature**. Disponível em:

<http://androiddevelopers.blogspot.in/2010/06/exercising-our-remote-application.html>. Acesso em 15 de abril. 2015.

[21] Google Inc. **Preparing for Release**. Disponível em:

<http://developer.android.com/tools/publishing/preparing.html>. Acesso em 15 de abril. 2015.

[22] Google Inc. **Managing Projects Overview**. Disponível em:

<http://developer.android.com/tools/projects/index.html#ApplicationProjects>. Acesso em 15 de abril. 2015.

[23] Google Inc. **Google Play In-app Billing**. Disponível em:

<http://developer.android.com/google/play/billing/index.html>. Acesso em 15 de abril. 2015.

[24] Google Inc. **Get Started with Publishing**. Disponível em:

<http://developer.android.com/distribute/googleplay/start.html>. Acesso em 15 de abril. 2015.

[25] MITRE Corp. **Common Vulnerabilities and Exposures**. Disponível em:

<http://cve.mitre.org/>. Acesso em 15 de abril. 2015.

[26] WIKIPEDIA. **Transaction Authentication Number**. Disponível em:

<http://en.wikipedia.org/wiki/Transaction_authentication_number>. Acesso em 15 de abril. 2015.

[27] ANDRE G.; RAMOS P. . **BOXER SMS Trojan**. ESET Latin American Lab, 2013.

[28] Google Inc. **Dashboards**. Disponível em:

<https://developer.android.com/about/dashboards/index.html?utm_source=suzunone>. Acesso em 15 de abril. 2015.

[29] **Android Fragmentation**. Disponível em:

<http://pt.opensignal.com/reports/2014/android-fragmentation/>. Acesso em 15 de abril. 2015.

[30] MITRE Corp. **Android Security Vulnerabilities**. Disponível em:

<http://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-19997/Google-Android.html>. Acesso em 15 de abril. 2015.

[31] Bluebox Labs. **Santa or the Grinch: Android Tablet Analysis**. Disponível em:

<https://bluebox.com/business/santa-or-the-grinch-android-tablet-analysis-2014/>. Acesso em 15 de abril. 2015.

[32] WIKIPEDIA. **Android Version History**. Disponível em:

<http://en.wikipedia.org/wiki/Android version history>. Acesso em 15 de abril. 2015.

[33] XING L.; PAN X.; WANG R.; YUAN K.; WANG X. . **Upgrading your android, elevating my malware: Privilege escalation through mobile os updating**. IEEE, 2014.

[34] **Android Trickery**. Disponível em:

<http://c-skills.blogspot.com/2010/07/androidtrickery.html>. Acesso em 15 de abril. 2015.

[35] YAJIN Z.; XUXIAN J. . **Dissecting Android Malware: Characterization and Evolution**. IEEE, 2012.

[36] **Rage against the Cage**. Disponível em:

<https://github.com/bibanon/androiddevelopment-codex/blob/master/General/Rooting/rageagainstthecage.md>. Acesso em 15 de abril. 2015.

[37] **GingerBreak**. Disponível em:

<http://forum.xda-developers.com/showthread.php?t=1044765>. Acesso em 15 de abril. 2015.

[38] **z4Root**. Disponível em:

<https://github.com/bibanon/android-developmentcodex/blob/master/General/Rooting/z4root.md>. Acesso em 15 de abril. 2015.

[39] Bluebox Labs. **Android Security Analysis Challenge: Tampering Dalvik Bytecode During Runtime**. Disponível em:

<http://bluebox.com/labs/android-security-challenge/>. Acesso em 15 de abril. 2015.

[40] ZHOU Y.; JIANG X. . **Android Malware Genome Project**. IEEE, 2012.

[41] CASTILLO C. A. . **Android Malware Past, Present, and Future**. Mobile Working Security Group McAfee, 2012.

[42] Lookout Inc. **State of Mobile Security**. Lookout Mobile Security, 2012.

[43] Lookout Inc. **Current World of Mobile Threats**. Lookout Mobile Security, 2013.

[44] LEVER C.; ANTONAKAKIS M.; REAVES B.; TRAYNOR P.; LEE W. . **The Core of the Matter: Analyzing Malicious Traffic in Cellular Carriers**. NDSS, 2013.

[45] H. T. T. Truong, E. Lagerspetz, P. Nurmi, A. J. Oliner, S. Tarkoma, N. Asokan, S. Bhattacharya. **The Company You Keep: Mobile Malware Infection Rates and Inexpensive Risk Indicators**. USA, 2014.

[46] **Carat: Collaborative Energy Diagnosis**. Disponível em:

<http://carat.cs.berkeley.edu/>. Acesso em 15 de abril. 2015.

[47] **Fake NetFlix - Android trojan info stealer**. Disponível em:

<http://contagiominidump.blogspot.in/2011/10/fake-netxflix-adtroidtrojan-info.html>. Acesso em 15 de abril. 2015.

[48] SHAHZAD F.; AKBAR M. A.; FAROOQ M. . **A Survey on recent advances in malicious applications Analysis and Detection techniques for Smartphones**. Pakistan, 2012.

[49] **Top 10 Android Security Risks**. Disponível em:

<http://www.esecurityplanet.com/views/article.php/3928646/Top-10-Android-Security-Risks.htm>. Acesso em 15 de abril. 2015.

[50] **Spitmo vs Zitmo: Banking Trojans Target Android**. Disponível em:

<https://blogs.mcafee.com/mcafee-labs/spitmo-vs-zitmo-bankingtrojans-target-android>. Acesso em 15 de abril. 2015.

[51] **Backdoor.AndroidOS.Obad.a**. Disponível em:

<http://contagiominidump.blogspot.in/2013/06/backdoorandroidosobada.html>. Acesso em 15 de abril. 2015.

[52] **Fakedefender.B - Android Fake Antivirus**. Disponível em:

<http://contagiominidump.blogspot.in/2013/11/fakedefenderb-androidfake-antivirus.html>. Acesso em 15 de abril. 2015.

[53] **avast! Free Mobile Security**. Disponível em:

<http://www.avast.com/freemobile-security-c?utm expid=2275583821.bXJmQHnQA6pakUW6PaLQQ.2&utm referrer=https%3A%2F%2Fwww.google.com%2F>. Acesso em 15 de abril. 2015.

[54] **Reverse Engineering with ApkTool**. Disponível em:

<https://code.google.com/android/apk-tool>. Acesso em 15 de abril. 2015.

[55] Google Inc. **Class to Dex Conversion with Dx**. Disponível em:

<http://developer.android. com/tools/help/index.html>. Acesso em 15 de abril. 2015.

[56] ZHOU W.; ZHOU Y.; JIANG X.; NING P. . **Detecting Repackaged Smartphone Applications in Third-party Android Marketplaces**. USA, 2012.

[57] **Remote Access Tool Takes Aim with Android APK Binder**. Disponível em:

<http://www.symantec.com/connect/blogs/remote-access-tool-takes-aimandroid-apk-binder>. Acesso em 15 de abril. 2015.

[58] **Android/NotCompatible Looks Like Piece of PC Botnet**. Disponível em:

<http://blogs.mcafee.com/mcafee-labs/androidnotcompatible-lookslike-piece-of-pc-botnet>. Acesso em 15 de abril. 2015.

[59] RASTOGI V.; CHEN Y.; JIANG X. . **Droidchameleon: Evaluating Android anti-malware against Transformation attacks**. ACM, 2013.

[60] ZHENG M.; LEE P. P. C.; LUI J. C. S. . **ADAM: An Automatic and Extensible Platform to Stress Test Android Anti-virus Systems**. DIMVA, 2012.

[61] **ProGuard**. Disponível em:

<http://proguard.sourceforge.net/>. Acesso em 15 de abril. 2015.

[62] **DexGuard**. Disponível em:

<http://www.saikoa.com/dexguard> . Acesso em 15 de abril. 2015.

[63] FARUKI P.; BHARMAL A.; LAXMI V.; GAUR M.; CONTI M.; MUTTUKRISHNAN R. . **Evaluation of android anti malware techniques against dalvik bytecode obfuscation**. IEEE, 2014.

[64] Dex Labs. **Dalvik Bytecode Obfuscation on Android**. Disponível em:

<https://dexlabs.org/blog/bytecode-obfuscation>. Acesso em 15 de abril. 2015.

[65] **Reverse Engineering with Androguard**. Disponível em:

<https://code.google.com/p/androguard/>. Acesso em 15 de abril. 2015.

[66] Google Inc. **Security Enchancements**. Disponível em:

<https://source.android.com/devices/tech/security/enhancements/index.html>. Acesso em 15 de abril. 2015.

[67] CONTI M.; CRISPO B.; FERNANDES E.; ZHAUNIAROVICH Y. . **CRêPE: A system for enforcing fine-grained context-related policies on android**. IEEE, 2012.

[68] NAUMAN M.; KHAN S.; ZHANG X. . **Apex: extending android permission model and enforcement with user-defined runtime constraints**. ACM, 2010.

[69] BUGIEL S.; DAVI L.; DMITRIENKO A.; FISCHER T.; SADEGHI A.-R. . **Xmandroid: A new android evolution to mitigate privilege escalation attacks**. Germany, 2011.

[70] ONGTANG M.; MCLAUGHLIN S. E.; ENCK W.; MCDANIEL P. D. . **Semantically rich application-centric security in android**. IEEE, 2009.

[71] **Reverse Engineering with Smali/Baksmali**. Disponível em:

<https://code. google.com/smali>. Acesso em 15 de abril. 2015.

[72] **Android Decompiling with Dex2jar**. Disponível em:

<http://code.google.com/p/dex2jar/>. Acesso em 15 de abril. 2015.

[73] **DARE: Dalvik Retargeting**. Disponível em:

<http://siis.cse.psu.edu/dare/>. Acesso em 15 de abril. 2015.

[74] OCTEAU D.; JHA S.; MCDANIEL P. . **Retargeting Android applications to Java bytecode**. ACM, 2012.

[75] **Dedexer**. Disponível em:

<http://dedexer.sourceforge.net/>. Acesso em 15 de abril. 2015.

[76] **JEB Decompiler**. Disponível em:

<http://www.android-decompiler.com/>. Acesso em 15 de abril. 2015.

[77] **Similarities for Fun & Profit**. Disponível em:

<http://phrack.org/issues/68/15.html>. Acesso em 15 de abril. 2015.

[78] **Google Bouncer: Bad guys may have an app for that**. Disponível em:

<http://www.techrepublic.com/blog/it-security/google-bouncer-badguys-may-have-an-app-for-that/7422/>. Acesso em 15 de abril. 2015.

[79] KORNBLUM J. . **Identifying almost Identical Files using Context Triggered Piecewise Hashing**. Digital Investigation 3, 2006.

[80] Blueblox Labs. **Android User Security Guide**. Disponível em:

<https://bluebox.com/android-user-security-guide/>. Acesso em 15 de abril. 2015.