
**UFRPE**

**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**TRABALHO DE CONCLUSÃO DE CURSO**

# DETECÇÃO DE APLICATIVOS MALICIOSOS NO SISTEMA OPERACIONAL ANDROID POR MEIO DE ANÁLISE ESTÁTICA AUTOMATIZADA

DIÓGENES JOSÉ CARVALHO DA SILVA

RECIFE – PE

09/2017

DIÓGENES JOSÉ CARVALHO DA SILVA

# DETECÇÃO DE APLICATIVOS MALICIOSOS NO SISTEMA OPERACIONAL ANDROID POR MEIO DE ANÁLISE ESTÁTICA AUTOMATIZADA

Trabalho de Conclusão de Curso apresentado ao Bacharelado em Ciência da Computação, da Universidade Federal Rural de Pernambuco, como parte dos requisitos necessários à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Fernando Antônio Aires Lins

RECIFE – PE

09/2017

**MINISTÉRIO DA EDUCAÇÃO E DO DESPORTO**
**UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO (UFRPE)**
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

http://www.bcc.ufrpe.br

## FICHA DE APROVAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO

Trabalho defendido por Diógenes José Carvalho da Silva às 14 horas do dia 06 de setembro de 2017, na sala 02 do CEAGRI-02, como requisito para conclusão do curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco, intitulado **Detecção de Malwares em Aplicativos Android Através de Análise Estática Automatizada e Aprendizagem de Máquina**, orientado por Fernando Antonio Aires Lins e aprovado pela seguinte banca examinadora:

Fernando Antonio Aires Lins
DEINFO/UFRPE

Robson Wagner Albuquerque de Medeiros
DEINFO/UFRPE

Danilo Ricardo Barbosa de Araújo
DEINFO/UFRPE

## RESUMO

A plataforma de aplicações móveis Android proporciona um ambiente de desenvolvimento amplo e aberto a vários tipos de software, porém essa liberdade acarreta possíveis vulnerabilidades de sistema que são infelizmente utilizadas para ataques de segurança. Entre elas, as vulnerabilidades no software e hardware que possibilitam a criação de ameaças à segurança do usuário como: *spywares*, *malwares* em seus diversos tipos e os *ransomwares*. Portanto, é necessário avaliar os aplicativos em busca dessas ameaças que estão crescendo em quantidade e complexidade. Para isso este trabalho tem como objetivo criar uma abordagem integrada com intuito de detectar possíveis vulnerabilidades em aplicativos desenvolvidos para dispositivos móveis Android. Esta abordagem é composta por outras duas estratégias, a saber: análise estática automatizada e identificação de perfis de aplicativos que contém ameaças por meio de metadados sobre eles. Contando com técnicas como *Web crawling* de lojas de aplicativos e coleta manual, foi gerada uma base de dados com 1000 aplicativos, sendo 500 infectados e 500 não infectados utilizando técnicas de *supersampling*, processos de extração e seleção de atributos de classificação tais como: TF-IDF, quantidade de ocorrência de termos, conversão de termos nominais para binários e normalização. Utilizando a base de dados criada para gerar modelos de classificação nos mais diversos algoritmos disponíveis no mercado, com o intuito de avaliar o presente trabalho, obteve-se métricas de precisão, falsos positivos e falsos negativos em taxas aceitáveis e comparáveis com trabalhos que apresentam as mesmas métricas como forma de avaliar os resultados encontrados.



Palavras-chave: detecção de ameaças de segurança em aplicativos Android, segurança em dispositivos móveis, análise estática em aplicativos Android

## ABSTRACT

The mobile applications platform known as Android provides a wide an open environment of application development to all kinds of software, however this freedom can bring possible software security vulnerabilities that can be used unfortunately to create threats to the operation system. There are vulnerabilities that comes from software and hardware that allows the creation of threats called: spyware, diverse kinds of malware, and with raising popularity, the ransomware. In this case is necessary to build application analysis to find out threats that are increasing in size and complexity. To accomplish this task, this research proposes a technique that combines multiple strategies to orchestrate a new technique that can detect threats and vulnerabilities inside applications developed to the Android mobile operational system. The strategy combines automatic static analysis and threat profile identification by metadata from an external source. Using techniques like web crawling to collect metadata from application stores, we generated a data set with 1000 applications, which 500 are infected and 500 aren't, using balancing technique such as super sampling, extraction and selection of features like: TF-IDF, frequency of terms, feature conversion from nominal to binary and normalization. Using the generated data set to create classification models with the most used machine learning algorithms used by other researchers, we could provide precision metrics, false positives, and false negatives at acceptable rates, comparable to other researches that presents the same performance metrics.

Key-words: detection of security threats on Android applications, mobile devices security, static analysis in Android applications

# LISTA DE FIGURAS

# LISTA DE QUADROS

# SUMÁRIO

# 1 INTRODUÇÃO

O modelo de negócio e distribuição de *software* nas plataformas móveis na última década se caracteriza pela possibilidade de tratar as aplicações de forma distribuída e com fácil acesso tanto para desenvolvimento quanto para utilização, com objetivo de agregar o maior número de usuários possíveis nos mais diferentes níveis de capacidade de hardware por meio de dispositivos diversos, como smartphones, tablets, notebooks, dentre outros.

Neste contexto, o sistema operacional para dispositivos móveis com maior apelo comercial e base de usuários é o Android (Statista, 2017). Sua arquitetura traz consigo possibilidades de atender à demanda de *software* que precisa lidar com heterogenia de dispositivos possibilitando a cada fabricante de hardware adaptar o mesmo para seu fim específico, mantendo a compatibilidade com aplicativos desenvolvidos previamente. Porém essas vantagens para o usuário e desenvolvedor acabam se refletindo em possíveis vulnerabilidades nos *softwares*, que são constantemente utilizadas para fins nocivos para o usuário do aparelho móvel.

Pela natureza da interface das aplicações móveis, mais amigáveis, valorizando a entrada de novos consumidores com uma curva de aprendizado reduzida e uma gama de aplicações, esse mercado já conseguiu ultrapassar em quantidade de usuários que acessam a internet o número de sistemas desktop que realizam a mesma tarefa (Statista; TECH *Android Poised to Knock Windows off Internet Perch*, 2017). Levando em consideração a proporção da base de usuários e o tipo de usuário que utiliza a plataforma, é possível alegar que existe uma necessidade crescente por segurança nestes dispositivos.

## 1.1 MOTIVAÇÃO

O sistema operacional Android é o mais utilizado no mundo considerando ambientes móveis (Statista, 2017). O mercado de dispositivos móveis, mais precisamente *smartphones*, está expandindo constantemente a cada ano. No entanto as ameaças aos usuários desse meio de informação estão aumentando em quantidade e complexidade na mesma velocidade que são desenvolvidas ferramentas que as identifique.

Para contextualizar esse fato foi identificado durante o ano de 2013 houveram 2,47 milhões de malwares em ambientes móveis de um total de 3,73 milhões de acordo com McAfee (MUÑOZ, Alfonso et al, 2015).

Observando o aumento de ameaças de segurança aos usuários na plataforma Android é preciso conhecer sua arquitetura, estrutura de desenvolvimento e distribuição de aplicativos para isso é necessário criar ferramentas de análise nas camadas mais críticas do sistema Android.

O usuário típico do sistema Android, em geral, não possui conhecimento técnico sobre segurança e, principalmente em países emergentes como o Brasil, utiliza fontes gratuitas e não oficiais para busca de aplicativos. Sendo assim, esse usuário contribui ativamente com a expansão de ameaças contidas nas aplicações que utiliza.

Observando o aumento de dispositivos Android nos últimos anos, as ameaças para esses dispositivos crescem proporcionalmente. Tendo em vista que o processo de publicação, aprovação e análise de aplicativos Android na loja *Play Store* é quase que totalmente automático, e que os aplicativos são passíveis de serem adicionados por meio de fontes não confiáveis, é interessante ter ferramentas de terceiros que contribuam para manter as bases de aplicativos oficiais e não oficiais livres de ameaças.

Análises técnicas manuais não conseguem identificar as ameaças de forma eficaz, devido à grande quantidade de aplicativos que são publicados diariamente em adição à diversos repositórios de terceiros que disponibilizam aplicativos de diversas fontes potencialmente não confiáveis. Mesmo as análises automáticas, que replicam o comportamento de dispositivos reais para executar ameaças e tentar prevenir criando uma base de dados comunitária, não conseguem alcançar todos os tipos de problemas existentes. Por isso, é relevante a criação e atualização de soluções, com licença de uso de código aberto entre os desenvolvedores para estimular a contribuição de software que pode ser utilizado na proteção, detecção e monitoramento de ameaças. Outra característica do ambiente geral de desenvolvimento de soluções de segurança para esse setor que complica a sua evolução é o fato de que o sistema Android está em constante modificação e se trata

de um projeto de código aberto e privado ao mesmo tempo, já que as empresas que desenvolvem os dispositivos inserem código proprietário no projeto da Google.

Então, sendo esse o cenário de ferramentas e plataformas de análise de segurança em dispositivos Android disponíveis hoje, é necessário buscar técnicas para melhorar a detecção e classificação de ameaças ou até mesmo reinventar as que já existem, pois, as ameaças de segurança tendem a ficarem cada vez mais difíceis de detectar.

## 1.2 OBJETIVOS

Nas duas seções seguintes os objetivos são explanados em geral e específicos.

### 1.2.1 GERAL

Este trabalho tem como objetivo desenvolver uma estratégia de detecção de ameaças de segurança em aplicativos do sistema operacional Android utilizando abordagens combinadas de análise estática no aplicativo a partir do seu código fonte e informações subjetivas encontradas em lojas de aplicativos.

Os objetivos específicos são descritos nos seguintes tópicos:

- Propor e desenvolver técnica de detecção de ameaças de segurança em aplicativos Android;
- Detectar ameaças de segurança em aplicativos Android;
- Identificar se a combinação das abordagens utilizadas é eficaz na detecção de ameaças de segurança em aplicativos Android;

## 1.3 ESTRUTURA DO DOCUMENTO

O restante deste documento está dividido da forma que se segue.

O capítulo 2 ilustra os conceitos da plataforma Android que sustenta as aplicações que este trabalho analisa, bem como as ameaças à segurança do sistema e também a teoria envolvida nas técnicas que são utilizadas para detectar tais ameaças.

Por sua vez, o capítulo 3 aborda os trabalhos relacionados que compõem técnicas similares ao da análise de aplicativos proposta neste trabalho, bem como seus pontos fortes e potenciais áreas de melhoria.

O capítulo 4 descreve a solução de análise estática automatizada que é proposta neste documento, utilizando os conhecimentos teorizados na seção de conceitos e a análise dos trabalhos similares citados no terceiro capítulo.

O capítulo 5 se propõe a avaliar os processos metodológicos propostos, demonstrar e aplicar processos de medição de desempenho da solução e comparar os resultados encontrados a partir da execução da mesma com alguns trabalhos que utilizam métricas de avaliação similares.

Por fim, o capítulo 6 é responsável por expor as conclusões sobre a abordagem proposta e suas implicações, acrescido de informações de possíveis melhorias na técnica.

## 1.4 ESCOPO DO TRABALHO

Este trabalho se propõe a analisar aplicativos que possuem ou não ameaças de segurança que são caracterizadas por alterações de *software* maliciosas, não se atendo a tipo de ameaça (*malware,ransonware,spyware*) e sim as estruturas de *software* que identificam uma ameaça dentro de um aplicativo do sistema operacional *Android*.

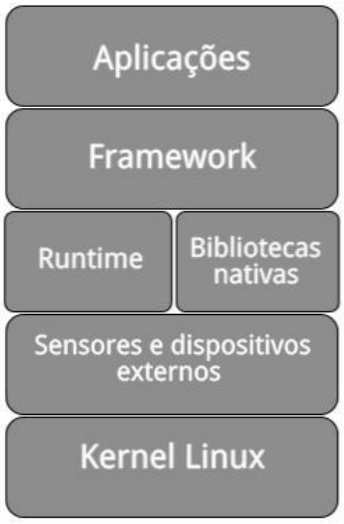
# 2 CONCEITOS

Este capítulo contém o conhecimento necessário para o entendimento dos processos utilizados e citados pela abordagem que este trabalho propõe, basicamente está apresentando uma breve descrição técnica da plataforma Android, dos riscos de segurança e das técnicas de detecção de ameaças que existem na literatura.

## 2.1 PLATAFORMA ANDROID

A plataforma é composta por uma pilha de sistemas que se comunicam de forma independente. Isso permite que essa estrutura seja extensível a qualquer dispositivo desenvolvido pelas empresas que utilizam o código fonte mantido pela Google.

Figura 2.1 Camadas do sistema Android.




Fonte: Produção do próprio autor.

Na Figura 2.1 a camada mais inferior que é um *kernel* Linux que, assim como outras distribuições dos sistemas operacionais que utilizam tal estrutura, carrega os *softwares* que realizam a comunicação dos dispositivos físicos, o sistema de gerenciamento de memória e energia.

A camada de sensores e dispositivos externos existe para criar uma interface de comunicação com o kernel para que as bibliotecas nativas e o *runtime* do sistema operacional em si possam utilizar dispositivos externos ao sistema nativo.

A camada *framework* é responsável por gerenciar as estruturas de desenvolvimento, por exemplo as *Activities* que representam uma execução de instância de um recurso do sistema e pode ser iniciado por um *Intent,* que é uma abstração da plataforma para o desenvolvedor utilizar os dispositivos físicos e aplicações de sistema, tais como: provedores de conteúdo, gerenciadores de recursos do dispositivo e sistema de exibição de imagem.

A camada de aplicações se caracteriza pela estrutura de software que comporta os aplicativos implementados por desenvolvedores utilizando as ferramentas disponíveis por meio da camada *framework.*

> A camada de aplicações é representada pelo invólucro dos aplicativos no seu formato de utilização: arquivos *Android PacKage* (APK); que nada mais são do que arquivos ZIP baseados nos arquivos JAR utilizados pela linguagem JAVA, usados no desenvolvimento de aplicativos para o sistema. Todos os aplicativos possuem meta informação armazenada dentro de um arquivo xml sobre permissões e arquitetura [...]. (BRAGA; DO NASCIMENTO; DA PALMA; ROSA, 2012)

Esta camada superior é a que vai ser mais focada neste trabalho, por ser o ponto mais acessível ao usuário, consequentemente a alguém que queira criar ameaças para benefício próprio e aqueles que querem identificar e prevenir modificações maliciosas nos aplicativos.

## 2.2 AMEAÇAS E VULNERABILIDADES

As ameaças podem se caracterizar em algumas categorias, sendo elas: *spywares*, *malwares* e *ransonwares*. Porém, como o *malware* é geralmente relacionado diretamente ao software da camada de aplicação e está relacionado às aplicações maliciosas, irá ser explanado com maior profundidade.

*Malware* é um dos tipos mais comuns de ameaça em dispositivos móveis e como já citado anteriormente, está presente em larga escala e por isso é considerado com frequência na área de segurança.

> Malware é o software que rouba, modifica, ou apaga dados de aplicações ou do sistema operacional do usuário sem o consentimento deste. Isto é, não foi obtida autorização prévia do usuário para a realização

> destas atividades. Em outras palavras, o software foi desonesto com o usuário [...]. (FELT, 2011 apud BRAGA; DO NASCIMENTO; DA PALMA; ROSA, 2012)

O comportamento do software nocivo conhecido como *malware* pode causar danos pessoais e monetários, e esses danos podem até passar despercebidos e serem irreversíveis, pela sua natureza de enganação do usuário por meio do disfarce em aplicações legítimas, por isso é uma ameaça que precisa ser constantemente identificada e erradicada.

## 2.3 TÉCNICAS PARA DETECÇÃO DE AMEAÇAS E VULNERABILIDADES

Duas possibilidades amplamente utilizadas para a de detecção e classificação de malwares são as técnicas: Análise Estática e Análise Dinâmica.

A análise estática se caracteriza pela verificação do código do software em seu estado de texto, antes de ser compilado, aplicando técnicas de análise de palavras por exemplo, podendo utilizar aprendizagem de máquina de forma automática para identificar perfis de código malicioso ou simplesmente análise manual a partir da leitura do código fonte em busca de partes que possam estar explorando alguma vulnerabilidade de sistema para criar algum tipo de ameaça (RIBEIRO, 2015).

Análise dinâmica é o método de emular o sistema operacional ou aplicação, monitorar seu funcionamento por padrões de comportamento suspeitos, tais como chamadas de sistema incomuns para o tipo de *software* (RIBEIRO, 2015).

Também é comum unir as duas abordagens para aumentar a cobertura de código analisado com monitoramento em execução, e esta técnica é chamada de híbrida. A técnica híbrida pode se beneficiar da verificação de permissões do *software* para diminuir a quantidade de código a ser monitorado, aumentando a eficiência da estratégia de detecção.

Tendo em vista o escopo deste trabalho, focado em análise estática, na próxima seção será explorada de forma mais detalhada a técnica de análise estática.

## 2.4 ANÁLISE ESTÁTICA

Esse tipo de análise consiste na adoção de técnicas e ferramentas em código pré-compilado, obtido por meio do código fonte da aplicação ou através da descompilação dos aplicativos.

Ferramentas de análise estática se encaixam nas seguintes categorias de atuação (NEUNER; VAN DER VEEN; LINDORFER; HUBER; MERZDOVNIK; MULAZZANI; WEIPPL, 2014):

**Extração de meta informação interna:** Extrair informação de manifestos de aplicativos e recuperar dados sobre permissões requisitadas e estruturas de desenvolvimento utilizadas (*Intents* e *Activities*).

Figura 2.2 Transformação de código fonte Java em Dex


Fonte: (STRAZZERE, 2012 apud BRAGA; DO NASCIMENTO; DA PALMA; ROSA, 2012)

**Descompilação:** Engenharia reversa do processo de compilação de aplicativos descrito na Figura 2.2, basicamente obtendo arquivos JAVA a partir de arquivos DEX.

## 2.5 META INFORMAÇÃO DE LOJAS DE APLICATIVOS

Nas lojas de aplicativos é possível encontrar informações sobre o aplicativo que não necessariamente estão relacionadas ao seu conteúdo, mas à sua capacidade, qualidade e procedência.

> Trabalhos prévios dentro dessa categoria de verificação de código (análise estática) são tradicionalmente voltados para áreas ligadas às características intrínsecas de aplicações Android, tais como as já citadas que criam atributos de perfil para *malwares* baseadas em permissões requisitadas pela aplicação, chamadas de funções específicas dentro do código. No entanto, a Play Store ou lojas de aplicativos Android em geral são uma fonte de informação que pode ser usada para identificar se a aplicação está infectada ou não. Já foi utilizada análise de sentimento em comentários como estratégia, bem como utilizar atributos selecionados a

> partir de dados criados sobre a aplicação na loja. (MUÑOZ, Alfonso *et al*, 2015, tradução nossa)

Como é possível observar no trabalho citado, utilizar fontes de informações externas ao aplicativo a ser analisado pode ser benéfico para criar um perfil de detecção de ameaças, pois geralmente é possível extrair conhecimento na forma de escrita do texto, intenção das palavras, análise de sentimento e estratégias de validação de conteúdo com a aplicação relacionada.

## 2.6 CONSIDERAÇÕES FINAIS

Este capítulo teve como objetivo possibilitar o entendimento de conceitos fundamentais associados a solução proposta neste trabalho. No próximo capítulo, os trabalhos relacionados a temática deste documento serão explicitados.

# 3 TRABALHOS RELACIONADOS

As abordagens utilizadas recentemente para lidar com detecção e classificação de *malwares* seguem geralmente um fluxo lógico comum descritos como análise estática, dinâmica ou híbrida. Os trabalhos geralmente tiram proveito de conhecimento prévio de outras abordagens já utilizadas por outros pesquisadores da área ou ferramentas que realizam parte do trabalho da análise para montar novas estratégias.

Geralmente, em um sistema completo, é aplicada uma ou duas grandes etapas distintas de análise, se não for possível separar o fluxo de cada etapa a estratégia é chamada de análise híbrida. O sistema IntelliDroid (WONG, Michelle Y.; LIE, David., 2016), demonstra em passos discretos esse conceito de etapas, onde há estratégias de análise dinâmica e análise estática, combinados posteriormente para gerar uma forma de identificar a condição do aplicativo infectado ou não. No entanto, ele não utiliza informações externas ao aplicativo, que podem adicionar melhores resultados de detecção de ameaças de segurança, somente dados internos, e também não traz estratégias de classificação com redes neurais ou classificadores em geral.

Existem abordagens que utilizam estratégias mais específicas em relação ao modelo de detecção, ao utilizar algoritmos de aprendizado de máquina (WEI, Te-En et al,2012) (LI, Qi; LI, Xiaoyu,2015) (YUAN, Zhenlong; LU, Yongqiang; XUE, Yibo,2016) (YERIMA, Suleiman Y.; SEZER, Sakir; MUTTIK, Igor, 2014, p. 37-42), (PEIRAVIAN, Naser; ZHU, Xingquan, 2013) (FARUKI, Parvez *et al*., 2013), com algoritmos tais como *Random Forests* (ALAM, Mohammed S.; VUONG, Son T, 2013), *Support Vector Machine* (SVM) (ZHAO, Min *et al*.,2011), Redes Bayesianas (YERIMA, Suleiman Y. *et al*, 2013), entre outros. Pode-se afirmar que estas abordagens têm nível aceitável de sucesso em encontrar padrões em atributos selecionados a partir de informação interna ao aplicativo, contudo elas não tratam informações externas sobre o aplicativo.

Outras abordagens para detecção de ameaças também já trataram a coleta de informações em lojas de aplicativo para gerar atributos de classificação de forma bem-sucedidas (ZHOU, Yajin *et al*, 2012), (ZHOU, Wu *et al*, 2012), (SANZ, Borja *et al*, 2013). Contudo, estas pesquisas não propõem encontrar alguma relação entre as

estratégias prévias de outras abordagens que incorporam classificação a partir de atributos internos ao aplicativo, e é possível afirmar que uma análise completa do aplicativo em si é de grande valia para uma técnica de detecção de ameaças.

Pelo fato de que uma análise completa da aplicação é benéfica na detecção de ameaças, existem pesquisas que abordam a similaridade entre aplicações infectadas, tais como (WU, Dong-Jie et al,2012), (SATO, Ryo; CHIBA, Daiki; GOTO, Shigeki, 2013), (CRUSSELL, Jonathan; GIBLER, Clint; CHEN, Hao, 2012), que tentam encontrar padrões maliciosos nas aplicações comparando com uma versão genuína da mesma ou tratando apenas de rastreio de chamadas de API dentro do aplicativo ou manifesto. Esse tipo de abordagem é útil para detecção de ameaças; porém, existem abordagens que tratam o aplicativo em execução ou em conjunto com alguma informação externa, e isso ajuda na detecção de certas famílias de *malwares*.

Ao se considerar a estratégia de detecção por aprendizagem de máquina, é preciso ter em mente dois passos importantes: seleção e extração de atributos. Por isso, os trabalhos citados a seguir têm relação com a abordagem a ser proposta por este documento, mais especificamente nas etapas que compõem a criação de um modelo de classificação.

Tendo essa informação anterior como ponto chave na criação de uma técnica eficiente, Shabtai (SHABTAI, Asaf; FLEDEL, Yuval; ELOVICI, Yuval, 2010) descreve uma técnica de classificação de aplicativos em categorias por meio de aprendizagem de máquina aponta que um número de no máximo 800 atributos selecionados em relação à uma base de dados na casa dos milhares se faz suficiente e limitante, para que não afete a performance da classificação.

A extração de atributos é uma etapa da criação de modelo de classificação que pode ter diversas formas de ser executada. Uma técnica que vários trabalhos, tal como (AAFER, Yousra; DU, Wenliang; YIN, Heng, 2013) utiliza é lidar com as chamadas de aplicação dentro do código fonte extraído dos aplicativos.

A seleção de atributos para um classificador criado utilizando dados de uma loja, a *Play Store*, já foi abordada por outro trabalho (MUÑOZ, Alfonso et al, 2015), que conseguiu listar 48 atributos dentre várias categorias de tratamento de informação:

- Informações do aplicativo;
- Informações de desenvolvedores;
- Informações de certificados;
- Informações de redes sociais;

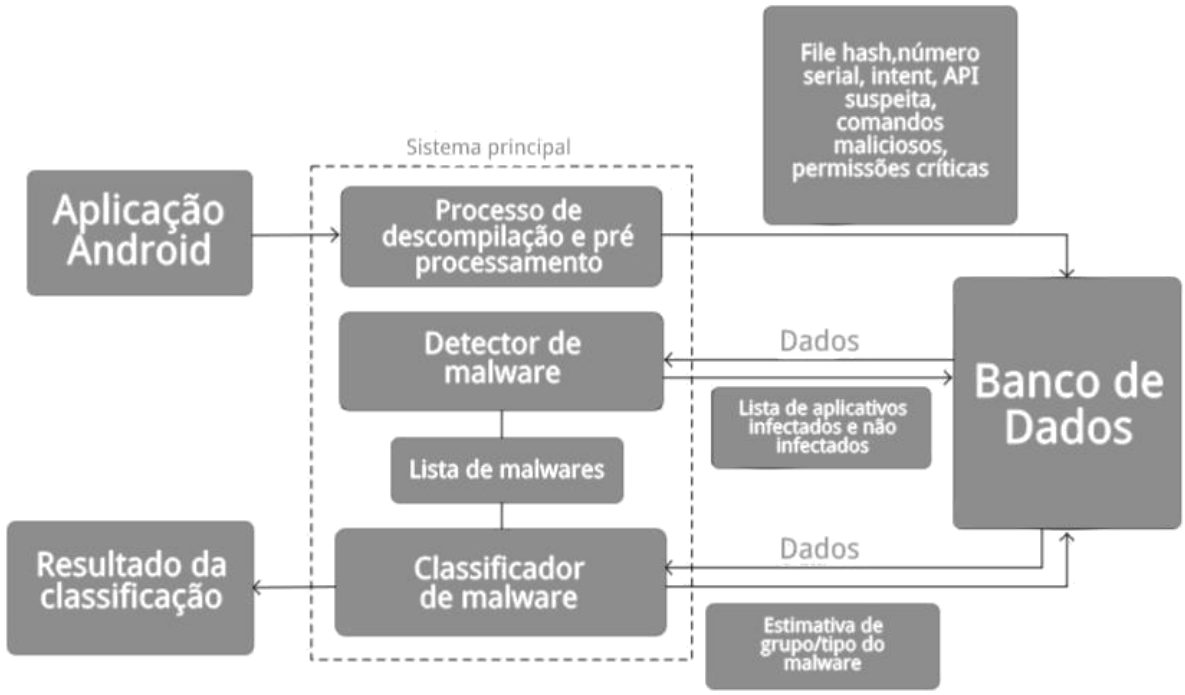
Porém os dados utilizados pelo trabalho do MUÑOZ não foram utilizados para classificar as ameaças, apenas detecta-las, assim como outro projeto (PEHLIVAN, Uğur *et al*, 2014).

Para extração de dados para atributos, *Web crawlers* é uma alternativa viável que foi utilizada por Apvrille (APVRILLE, Ludovic; APVRILLE, Axelle, 2015). Nesta iniciativa, foi realizado todo o processo de ordenação de atributos para melhor utilizar as informações obtidas e evitar falsos positivos. Entretanto, como conclusão, o autor fala que há como melhorar a seleção e extração de atributos mesmo utilizando dados oriundos de técnicas de análise dinâmica ou de outras fontes.

Um exemplo de sistema fechado e completo que trata de realizar a descompilação do aplicativo e utilizar informação de criador (KANG, Hyunjae et al,2015), que pode ser considerada uma abordagem de análise estática, em que o comportamento do desenvolvedor de *malware* é tratado como parâmetro, dentre eles, qual número de serial está na lista negra de desenvolvedores, comandos maliciosos mais usados em modo administrador, dentre outros dados. Esse processo gera a relação de aplicações classificadas de duas formas: se está ou não infectada e à qual tipo pertence. A Figura 3.1 mostra o fluxo de dados dessa abordagem, onde a partir da aplicação Android é realizado os processos de tratamento para permitir a extração de atributos de seleção e informações sobre o arquivo, a partir daí categorizados num banco de dados, que servirá para armazenar as características necessárias para detectar os *malwares*, lista-los, para assim ser possível classifica-los, estimando o grupo ou família a que pertence.

De acordo com os resultados dessa aplicação desenvolvida, tais como precisão e viabilidade, é possível notar que esse processo é válido no desenvolvimento de uma ferramenta prática e o artigo cita que para melhorias futuras são necessários mais atributos, detecção e classificação mais rebuscadas utilizando redes neurais diferentes ou composições de classificadores.

Figura 3.1 Sistema de análise estática com classificador.




Fonte: (KANG; JANG; MOHAISEN; KIM,2015, tradução nossa)

Dentro da discussão que este trabalho propõe, ficam claras as limitações das pesquisas anteriores por meio da variedade e quantidade de informações de entrada que poderiam ser utilizadas de forma combinada para contribuir com a qualidade de pesquisas futuras.

O Quadro 3.1 ilustra as técnicas utilizadas pelos trabalhos descritos nessa seção, e como é possível ver a maioria se beneficia das técnicas da análise estática, onde o processo é menos custoso e simplificado. É observavél também que o aprendizado de máquina se mostra consistente e eficiente na classificação de ameaças, sendo priorizado, porém a utilização de metadados de fontes externas à aplicação não aparece com frequência nos trabalhos.

Combinar as estratégias de análise e obtenção de fontes de dados para a detecção de ameaças pode ser benéfica tendo em vista que todas as pesquisas utilizadas abaixo utilizam pelo menos duas abordagens, assim como esse trabalho que se beneficia de três (Análise Estática, Aprendizagem de máquina e Metadados)

para propor uma nova técnica a ser explorada no capítulo 4. Além disso a estratégia proposta por este trabalho se diferencia das outras citadas por utilizar de metadados de lojas de aplicativos em combinação com análise estática coletados para a geração da base de treinamento utilizada na criação do modelo de classificação responsável por detectar as ameaças de segurança. Com a utilização dessa nova fonte de dados é possível aplicar a técnica a ser proposta para bases de dados de lojas de aplicativos não oficiais por exemplo, que buscam soluções de segurança que levem em consideração os dados usuários que as colocam no sistema.

Quadro 3.1 Comparativo de trabalhos relacionados

| Trabalho | Análise Estática | Análise Dinâmica | Aprendizagem de máquina | Metadados de fontes diversas | Metadados de lojas de aplicativos |
|---|---|---|---|---|---|
| IntelliDroid (WONG, Michelle Y.; LIE, David., 2016) | Sim | Sim | Não | Não | Não |
| (MUÑOZ, Alfonso et al, 2015) | Sim | Não | Sim | Sim | Não |
| (WEI, Te-En et al,2012) | Sim | Não | Sim | Não | Não |
| (LI, Qi; LI, Xiaoyu,2015) | Sim | Não | Sim | Não | Não |
| (KANG, Hyunjae et al,2015) | Sim | Não | Sim | Sim | Não |
| (YUAN, Zhenlong; LU, Yongqiang; XUE, Yibo,2016) | Sim | Não | Sim | Não | Não |
| (YERIMA, Suleiman Y.; SEZER, Sakir; MUTTIK, Igor, 2014) | Sim | Não | Sim | Não | Não |
| (PEIRAVIAN, Naser; ZHU, Xingquan, 2013) | Sim | Não | Sim | Não | Não |
| (FARUKI, Parvez et al,2013) | Sim | Não | Sim | Não | Não |
| (ALAM, Mohammed S.; VUONG, Son T,2013) | Sim | Não | Sim | Não | Não |
| (ZHAO, Min; GE, Fangbin; ZHANG, Tao; YUAN, Zhijian,2011) | Sim | Não | Sim | Não | Não |
| (YERIMA, Suleiman Y. et al, 2013) | Sim | Não | Sim | Não | Não |
| (ZHOU, Yajin et al,2012) | Sim | Sim | Sim | Não | Não |
| (ZHOU, Wu et al,2012) | Sim | Não | Não | Não | Não |
| (SANZ, Borja et al,2013) | Sim | Não | Sim | Não | Não |
| (WU, Dong-Jie et al, 2012) | Sim | Não | Sim | Não | Não |
| (SATO, Ryo; CHIBA, Daiki; GOTO, Shigeki, 2013) | Sim | Não | Sim | Não | Não |
| (CRUSSELL, Jonathan; GIBLER, Clint; CHEN, Hao, 2012) | Sim | Não | Não | Não | Não |
| (APVRILLE, Ludovic; APVRILLE, Axelle, 2015) | Sim | Não | Sim | Não | Não |
| (SHABTAI, Asaf; FLEDEL, Yuval; ELOVICI, Yuval, 2010) | Sim | Não | Sim | Não | Não |
| Proposta deste trabalho | Sim | Não | Sim | Sim | Sim |

## 3.1 CONSIDERAÇÕES FINAIS

Este capítulo objetivou descrever os trabalhos que estão na mesma área de atuação que a proposta de solução deste trabalho, que será descrita no próximo capítulo. Além disso, foi realizado um comparativo de como cada abordagem trata o processo de detecção de malwares ou ameaças em aplicações Android. No próximo capítulo, a solução proposta neste trabalho será detalhada.

# 4 TÉCNICA PARA DETECÇÃO DE APLICATIVOS MALICIOSOS NO SISTEMA OPERACIONAL ANDROID POR MEIO DE ANÁLISE ESTÁTICA AUTOMATIZADA

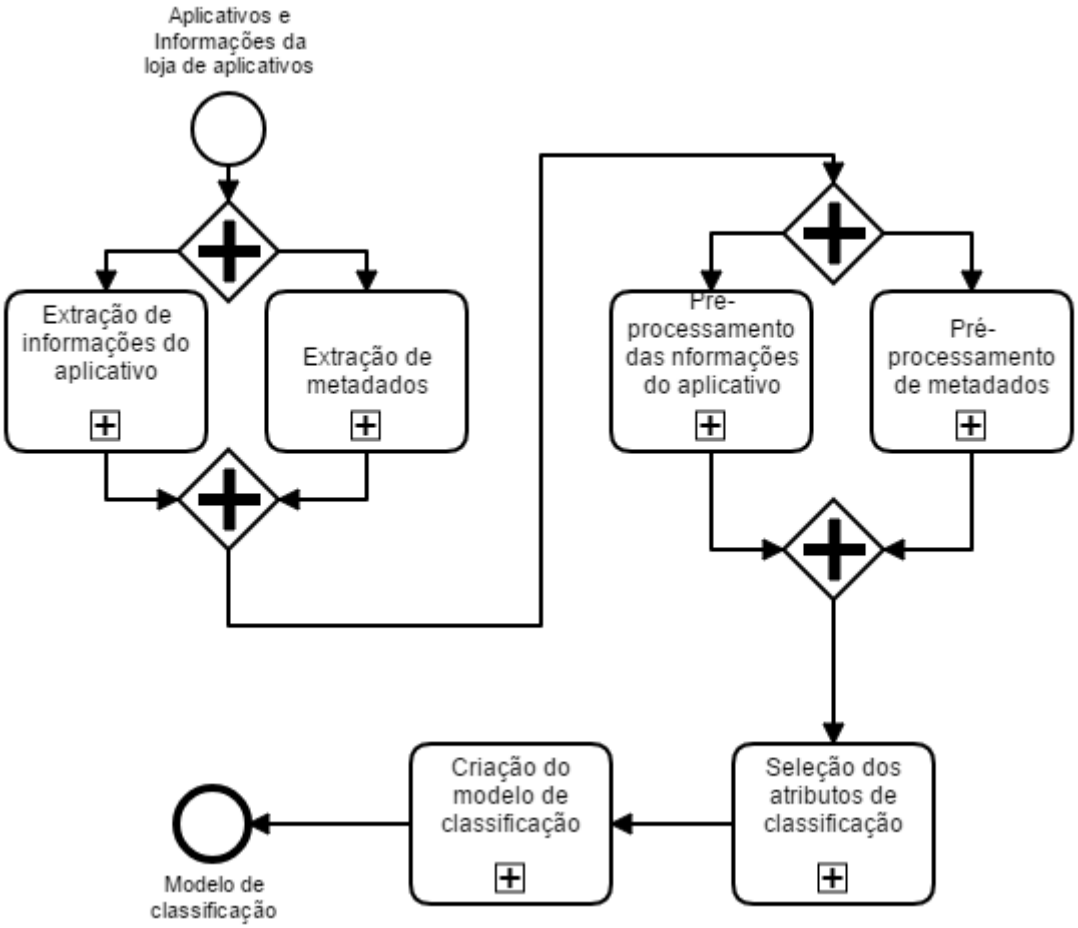
Este capítulo apresenta a contribuição principal deste trabalho, que é uma técnica para detectar ameaças de segurança em aplicativos instalados em dispositivos móveis que utilizam o sistema operacional Android. Esta solução consiste de etapas que descrevem como utilizar algoritmos de aprendizagem de máquina aliados a estratégias de extração de informações de duas fontes específicas, no caso uma é o arquivo da aplicação compactado e a outra é uma fonte de informações que evidenciem metadados do aplicativo.

A Seção 4.1 a ilustra o fluxo geral da técnica proposta por meio de um diagrama *Business Process Model and Notation* (BPMN), que é composto por subprocessos, que serão explicados no decorrer da Seção 4.1 e a arquitetura da solução na 4.2 que desenvolve a técnica e recursos de software envolvidos na execução da abordagem descrita a seguir.

## 4.1 VISÃO GERAL

A técnica proposta neste trabalho tem como entrada a aplicação Android em seu formato compactado APK e informações de uma loja de aplicações sobre o aplicativo disponíveis em uma página da Web, e apresenta como saída um modelo de classificação capaz de detectar se um aplicativo está infectado ou não. Uma visão geral da técnica é apresentada na Figura 4.1. O modelo de classificação gerado servirá na prática para identificar se uma aplicação está infectada ou não baseado no treinamento realizado com uma base de dados formatada para este objetivo.

Figura 4.1 Processo BPMN da técnica proposta.




Fonte: produção do próprio autor.

O fluxo identificado na Figura 4.1 se refere ao processo da solução proposta para a criação do modelo de classificação que pode ser utilizado para detectar ameaças de segurança em aplicativos e está dividido em subprocessos gerais, sendo eles quatro grandes etapas:

- **Extração**. Nesta etapa, os aplicativos e metadados são coletados e persistidos em um repositório de dados qualquer;
- **Pré-processamento.** Neste momento, os dados obtidos na etapa anterior são tratados com o objetivo de criar atributos de classificação para gerar uma base de treinamento;
- **Seleção de atributos.** Nesta etapa, é realizado o ranqueamento e a remoção de atributos com menor valor de classificação de acordo com o resultado do

ranqueamento de pesos do algoritmo utilizado, que irá ser citado posteriormente.

- **Criação de modelos de classificação**. Finalmente, neste momento, são gerados os modelos de classificação utilizando os algoritmos escolhidos, a partir do processo de treinamento, utilizando validação cruzada para identificar a eficácia dos modelos e a escolha do melhor de acordo com as métricas que irão ser detalhadas na subseção que representa essa etapa.

Os processos de extração e pré-processamento são independentes entre as duas fontes de informações utilizadas, podendo ser separados em duas etapas distintas que podem ser executadas em paralelo para garantir menor tempo de execução da técnica como um todo.

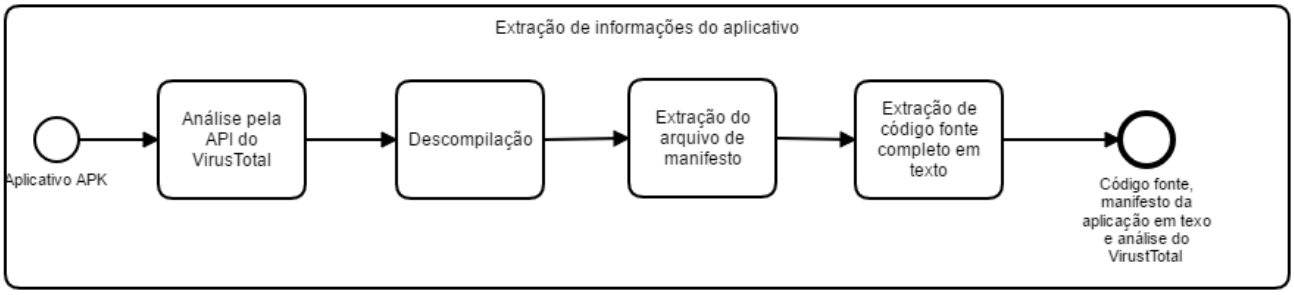
Com a introdução apresentada nesse início de capitulo as subseções explanarão cada subprocesso explicando seus objetivos, descrevendo as atividades que os compõem.

### 4.1.1 EXTRAÇÃO DE INFORMAÇÕES DOS APLICATIVOS

Este subprocesso ilustrado na Figura 4.2 é composto por etapas que definem a coleta de informações a partir do arquivo da aplicação.

Inicialmente existe as aplicações Android, com isso se faz necessária a definição de duas classes de controle (infectados e não infectados) para tornar possível a criação de modelos de classificação a partir da base de treinamento formada pelos *samples* categorizados por atributos criados com as informações coletadas, a serem explicados nas próximas seções que compõem a técnica.

Figura 4.2 Subprocesso de extração de informações dos aplicativos.




Fonte: produção do próprio autor.

O processo de criação das classes de controle e definição de aplicativos em cada uma define a primeira etapa do subprocesso descrito na figura 4.2 que é utilizada a API do site VirusTotal (VirusTotal, 2017) para se obter informações a partir dos aplicativos que forem identificados como infectados por pelo menos um dos antivírus que o VirusTotal utiliza para realizar a análise. Essa etapa pode ser realizada por outro sistema de análise de aplicativos, pois o objetivo é criar uma base de treinamento a partir do conhecimento obtido dos aplicativos infectados.

Posteriormente se faz necessário criar e utilizar *scripts* de descompilação, extração de manifesto e código que constituem nas três etapas que se seguem no esquema BPMN do subprocesso descrito acima. A implementação do subprocesso pode ser realizada com auxílio de ferramentas de terceiros.

Realizando a estratégia descrita nesse subprocesso, o banco de dados terá informações relevantes sobre o aplicativo persistidas, sendo elas:

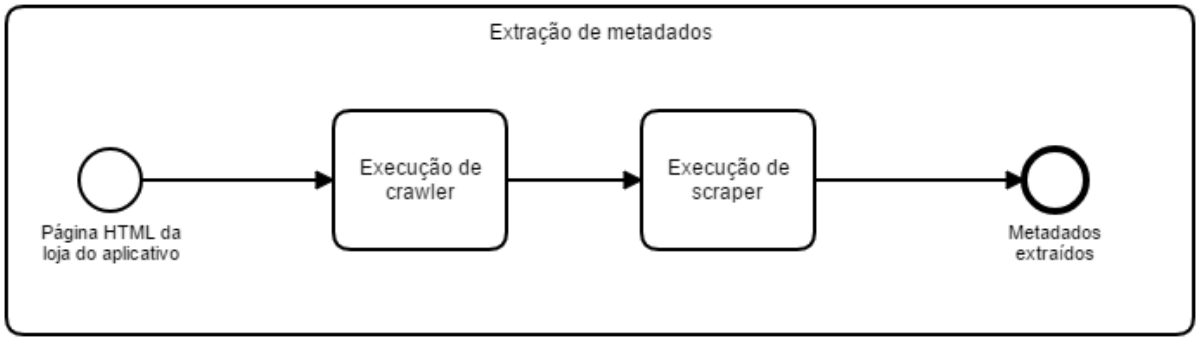
- Código da aplicação em formato de texto;
- Manifesto da aplicação em formato de texto;
- Análise de *malware* da aplicação do site VirusTotal contendo a verificação de *antivírus* dos mais diversos disponíveis no mercado, onde é possível identificar uma possível ameaça de segurança se foi detectada em uma quantidade razoável de antivírus que fica a critério da implementação.

Todas essas informações irão ser refinadas para possibilitar a criação de uma base de treinamento adequada.

### 4.1.2 EXTRAÇÃO DE METADADOS

O subprocesso descrito nesta subseção (e mostrado na Figura 4.3) tem como objetivo capturar metadados da aplicação a partir de uma fonte de dados que possuam determinadas informações específicas. No caso desta solução, foi escolhida a página *HyperText Markup Language* (HTML) da loja de aplicativos para retirar os metadados.

Figura 4.3 Subprocesso de extração de metadados.




Fonte: produção do próprio autor.

Com a página HTML da aplicação na loja de aplicativos é possível ter acesso a informações relevantes à aplicação, para isso foi utilizado um *crawler* e um *scraper*, que são técnicas de navegação e extração automática de informações a partir de requisições às páginas HTML, que coletaram as seguintes informações:

- Descrição;
- Data de publicação;
- Desenvolvedor;
- Número de *downloads*;
- Tamanho da aplicação;
- Categoria;
- Título;
- Versão do sistema operacional *Android* requerida;
- Imagens da aplicação;

Essas informações foram escolhidas por serem relacionadas diretamente com a aplicação que as descreve, com objetivo de capturar valor de correlação entre o aplicativo e os metadados.

A partir de uma série de tratamentos a serem detalhados na seção de pré-processamento, essas informações têm a capacidade de se tornar relevantes na identificação de um perfil de aplicativo mais provável de estar comprometido com algum *malware*.

O perfil pode ser montado baseado na ideia de que a aplicação infectada precisa ter algum atrativo para que usuários sem conhecimento baixem a mesma, então palavras da descrição, categoria, título e imagens podem apresentar um

padrão em comum de excesso de benefícios. Também é possível obter padrões de classificação por meio de uma análise de intenção das palavras da descrição, onde palavras associadas com uma abordagem de enganação podem ser identificadas. Por fim, restam os padrões mais técnicos onde o tamanho da aplicação pode sugerir que a mesma está infectada com base na ideia de que o mesmo tem mais código do que deveria. Já o número de downloads pode indicar que a aplicação foi recentemente inserida na loja e talvez não tenha recebido nenhum tipo de avaliação de antivírus ou seja, provavelmente foi ré-empacotada após inserção de um *malware*, que é uma abordagem comum entre criadores de ameaças, copiando aplicações legitimas, inserindo código malicioso, recompilando e publicando em uma loja de aplicativos.

### 4.1.3 PRÉ-PROCESSAMENTO

A etapa de pré-processamento visualiza as técnicas de formatação, limpeza e construção de atributos de classificação que compõem a base de treinamento.

Assim, os atributos devem ser normalizados, organizados e estruturados a partir das informações previamente coletadas.

Essa etapa é uma das mais importantes, pois é nela que é construída uma base de dados que fornece subsídios para quantificar e qualificar os dados que foram obtidos, diminuindo a quantidade de ruído, informações irrelevantes e componentes incompletos.

Para criar atributos de classificação, se faz necessário analisar e coletar o máximo de informações relevantes da aplicação, assim como tratar corretamente os dados da loja de aplicativos, removendo dados desnecessários ou que não agreguem valor de classificação.

Na abordagem descrita nesse trabalho, a etapa de pré-processamento foi especificamente dívida em dois subprocessos como visto na Figura 4.1, onde serão detalhados nas subseções 4.3.1 e 4.1.3.2.

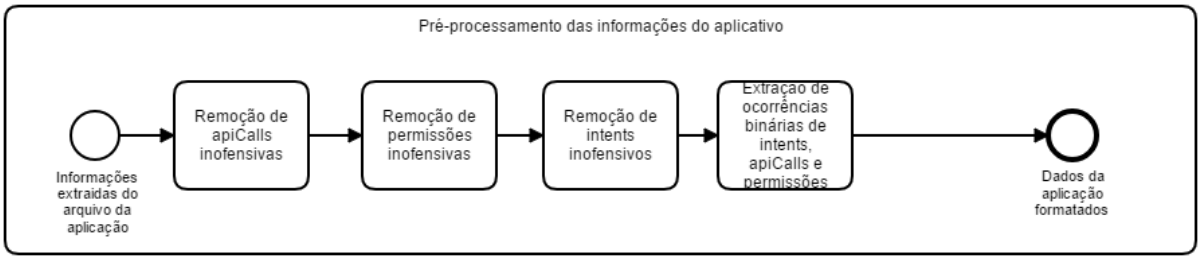
#### 4.1.3.1 PRÉ-PROCESSAMENTO DAS INFORMAÇÕES DOS APLICATIVOS

Este subprocesso de pré-processamento tem como objetivo tratar os dados recolhidos da aplicação em formato de texto e transforma-los em atributos de ocorrência binária, representados por o número 1, caso apareçam no texto ou o número 0, caso não apareçam no texto para adicionar valor quantitativo de classificação a partir de informações numéricas, que é importante pois os classificadores geralmente são formados por expressões matemáticas e controle de pesos, como redes neurais e SVM.

A partir do código da aplicação e manifesto coletados foi possível extrair as permissões que o aplicativo requisita ao usuário na instalação, *apiCalls* que são chamadas de função do sistema operacional para obter algum recurso, como envio de mensagens de texto, e *Intents* que são estruturas de controle do sistema que representam uma ação de sistema, como uma requisição de chamada telefônica. Essas três representações de estruturas de código foram tratadas como atributos relacionados à aplicação em formato de texto, para então realizar os processos de remoção de dados desnecessários e conversão em atributos numéricos.

As três primeiras atividades descritas no fluxo definido na Figura 4.4 possuem o mesmo objetivo e irão ser detalhadas agora.

Figura 4.4 Subprocesso de pré-processamento das informações dos aplicativos.




Fonte: produção do próprio autor.

O código da aplicação precisava ser tratado como uma lista de chamadas de funções de sistema (*apiCalls*), pois de acordo com trabalhos semelhantes, tal como (ZHENG, Min; SUN, Mingshen; LUI, John CS,2015) que utiliza como artefato de análise estatística para identificar assinaturas de malwares, essa é a informação mais relevante e de fácil acesso sobre o código fonte em texto. Porém nem todas as

chamadas de funções de sistema indicam alguma atividade maliciosa, por isso foram removidos os que são considerados inofensivos, pois não são invasivos aos dados ou processos de sistema, como o acesso à lanterna do dispositivo, que é uma *ApiCall* de baixo nível e crítica no sistema operacional, porém não fornece dados sensíveis ou causa danos ao dispositivo e usuário.

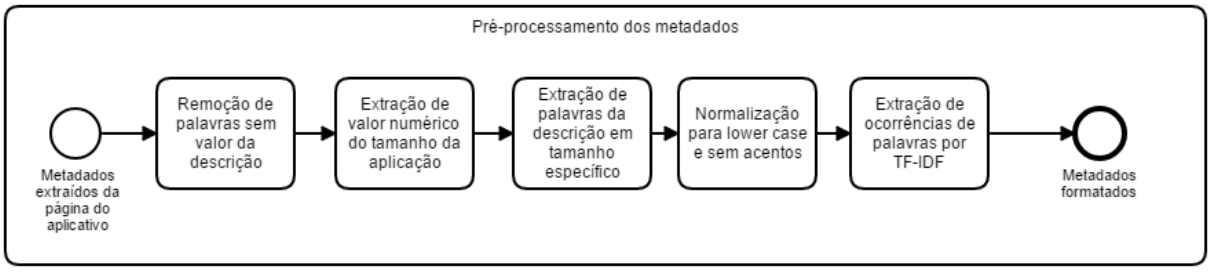
As permissões e *Intents* descritos no manifesto da aplicação foram utilizadas como atributos de classificação, para isso foi necessário adicionar no banco de dados todas as permissões que cada aplicativo descreve como texto para posteriormente criar atributos baseados nesse texto, então foram removidas as ocorrências de permissões desnecessárias que não podem ser associadas à utilização dos dados do usuário ou do dispositivo.

#### 4.1.3.2 PRÉ-PROCESSAMENTO DOS METADADOS

Este subprocesso de pré-processamento se propõe a tratar os metadados coletados a partir da página HTML onde a aplicação pode ser obtida, utilizando técnicas de extração de informação em texto para gerar atributos numéricos baseados na ocorrência das palavras com maior valor de classificação.

Este subprocesso está ilustrado na Figura 4.5, e logo em seguida cada uma dessas atividades será detalhada.

Figura 4.5 Subprocesso de pré-processamento de metadados.




Fonte: produção do próprio autor.

A descrição da aplicação exerce papel importante na geração de atributos, então foi necessário remover termos desnecessários como *stop words*, que são palavras irrelevantes para algoritmos de processamento de texto, pois representam apenas ligação semântica para o leitor, exemplos: as; e; os; de; para; com; sem; foi; resultando em palavras que agregam valor de classificação refinado, concluindo o primeiro passo do subprocesso.

A segunda atividade descreve a conversão de texto do tamanho da aplicação que está no formato <número><identificador>, por exemplo: “18MB’’, devido à natureza da coleta das informações a partir de uma página da Web, em um formato de inteiro apenas com o número.

A terceira atividade objetiva a remoção de palavras com tamanho maior que 25 caracteres e menor que 4 caracteres, esse processo é importante pois palavras muito grandes não aparecem com frequência em textos e como estamos lidando com valor de frequência de palavras, acabam sendo apenas ruídos, bem como palavras muito pequenas, que se não são *stop words,* são apenas complementos e maneirismos linguísticos para o texto manter a sintaxe correta.

A quarta atividade remove acentos das palavras e coloca todas em formato minúsculo (*lower case*). Para extrair atributos da descrição do aplicativo foi utilizada a técnica *Term Frequency Inverse Document Frequency* (TF-IDF) (medida estatística de qualidade de termos em um texto que leva em consideração frequência de palavras, *Term Frequency*, e o inverso da frequência nos documentos que as contém, *Inverse Document Frequency*) com ranqueamento com cortes de 10% a 90% de frequência de palavras, sendo assim as palavras com maior relevância foram escolhidas e tratadas como atributos, possibilitando a remoção de termos irrelevantes por muita frequência e pouca frequência, concluindo a quarta atividade.

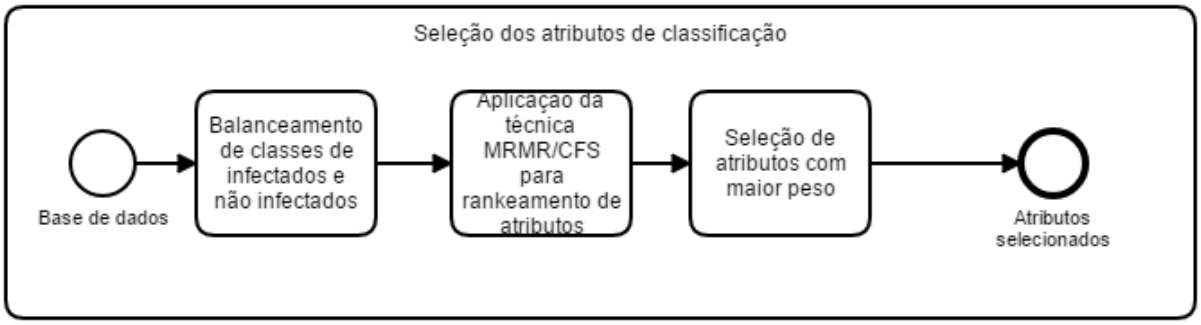
Por fim temos atributos de classificação bem estruturados, todos numéricos, porém em grande quantidade, na casa dos milhares, pois os textos são compostos de muitas palavras e cada palavra acaba se tornando um atributo, impossibilitando a utilização de classificadores sem causar *overfiting* (classificação com utilidade apenas para uma base de treinamento muito específica, sem flexibilidade) para isso se faz necessário selecionar os melhores atributos e a próxima seção descreve esse processo.

### 4.1.4 SELEÇÃO DOS ATRIBUTOS DE CLASSIFICAÇÃO

Este subprocesso definido na Figura 4.6 é responsável por balancear a base de dados com igual quantidade de aplicativos infectados e não infectados para que seja possível ter resultados de classificação que possam identificar ambas as

classes em precisão semelhante, pois é importante saber se aplicativos não estão infectados também, bem como aplicar técnica de ranqueamento de atributos e selecionar os melhores ranqueados.

Figura 4.6 Subprocesso de seleção dos atributos de classificação.




Fonte: produção do próprio autor.

A partir dos campos de texto das chamadas de aplicação, permissões, descrição da aplicação, tamanho da aplicação e categoria serão gerados atributos de classificação após a seleção. Alguns atributos foram descartados por estarem incompletos ou inconsistentes, sendo eles: número de downloads, categorias e versão do sistema operacional requerida.

Todas as amostras são normalizadas com o processo de normalização z-score, bem definido pela documentação do sistema MatLab (MathWorks, 2017) com o objetivo de ter atributos numéricos, para ser possível realizar modificações no tamanho da base de dados com maior facilidade, tendo em vista que a proporção de elementos de classes diferentes (infectado ou não) está desbalanceado por ser muito mais custoso encontrar aplicativos infectados em relação a não infectados.

Para melhorar o processo de classificação foi necessário a utilização da técnica de *supersampling*, que consiste na geração de novos *samples* pela utilização de similaridade matemática tendo como base os já existentes com o intuito de balancear as classes sem comprometer a base de treinamento, aumentando o número de casos de aplicativos infectados para um valor na casa das centenas, e selecionando centenas de casos de aplicativos não infectados na mesma proporção dessa forma a base de dados se torna balanceada. Esse processo descreve a primeira atividade do fluxo acima.

O processo de seleção de atributos, ilustrado como segunda etapa do processo acima, utiliza o algoritmo *Correlation Feature Selection* (CFS), responsável por identificar atributos com maior correlação com as classes, utilizando ranqueamento com parâmetros:

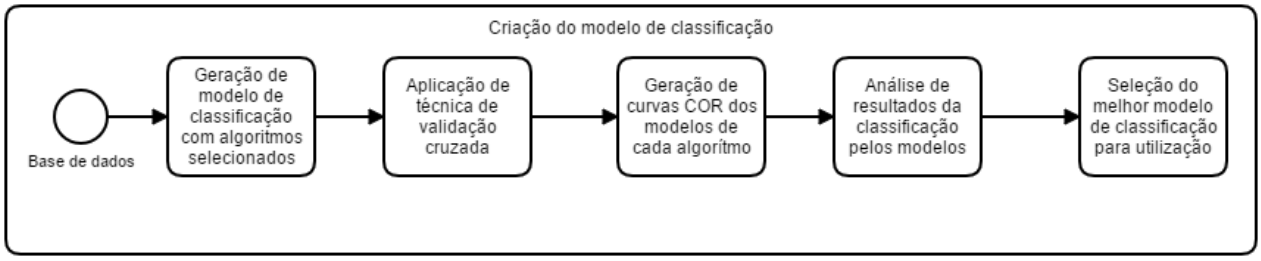
- Ordenação descendente, onde os cálculos de pesos dos atributos mais importantes são priorizados;
- Calculo de ranqueamento completo, comparando todos os atributos entre si, para melhor qualidade dos pesos calculados, as custas de maior tempo de execução;

A partir do ranqueamento de atributos são selecionados os atributos de classificação já mencionados de acordo com o peso resultante da técnica aplicada, sendo esta a última atividade. Concluindo a etapa de seleção é possível obter os atributos que serão utilizados para a criação do modelo de classificação.

### 4.1.5 CRIAÇÃO DO MODELO DE CLASSIFICAÇÃO

Este subprocesso de criação do modelo de classificação, representado na Figura 4.7, descreve a criação de modelos de classificação a partir da base de dados com os atributos selecionados, utilizando técnica de validação e gerando visualização de dados para a escolha do melhor modelo de classificação.

Figura 4.7 Subprocesso de criação do modelo de classificação




Fonte: produção do próprio autor.

O processo de geração de modelos de classificação é similar para todos os classificadores, a partir da base de treinamento e utilizando a técnica de validação cruzada com N partes para gerar resultados de precisão e matriz de confusão que

serão utilizados para identificar a qualidade do modelo gerado, descrevendo as duas primeiras atividades do fluxo da Figura 4.7.

Os algoritmos utilizados foram: *Random Forest*, SVM, *Deep Learning* e Árvore de decisão. Os parâmetros de cada algoritmo devem ser testados e escolhidos manualmente para se adequar melhor ao tipo de atributo da base de treinamento (numéricos), possibilitando melhor qualidade de classificação se melhores testados e aprovados com técnicas rigorosas, mas para critério de ilustração da proposta descrita nesse trabalho como prova de conceito, estão descritos nos quadros 4.1, 4.2, 4.3, 4.4, 4.5 e 4.6 para possibilitar a replicação da técnica de forma a atingir níveis aceitáveis nas métricas de avaliação.

Quadro 4.1 Parâmetros para Random Forest

| Algoritmo | Arvores | Índice de confiança | Profundidade máxima | Critério de avaliação |
|---|---|---|---|---|
| Random Forest | 100 | 0.5 | 150 | Ganho de informação |

Quadro 4.2 Parâmetros para SVM

| Algoritmo | Kernel | Estratégia | Cache | Tolerância no critério de avaliação |
|---|---|---|---|---|
| SVM | RBF | Nu-SVC 0.5 | 80 | 0.001 |

Quadro 4.3 Parâmetros para Deep Learning

| Algoritmo | Estratégia de avaliação | Nós | Épocas de treinamento | Estratégia de treinamento |
|---|---|---|---|---|
| Deep Learning | Tanh | 50 | 10 | Frequência adaptativa |

Quadro 4.4 Parâmetros para Arvore de decisão

| Algoritmo | Critério de avaliação | Profundidade máxima |
|---|---|---|
| Arvore de decisão | Ganho de informação | 250 |

#### 4.1.5.1 ANÁLISE E SELEÇÃO DO MODELO DE CLASSIFICAÇÃO

Após a criação dos modelos de classificação aplicando as bases de treinamento aos algoritmos citados anteriormente, é possível analisar o melhor,

sendo a penúltima etapa, levando em consideração os seguintes fatores que são resultantes da validação cruzada:

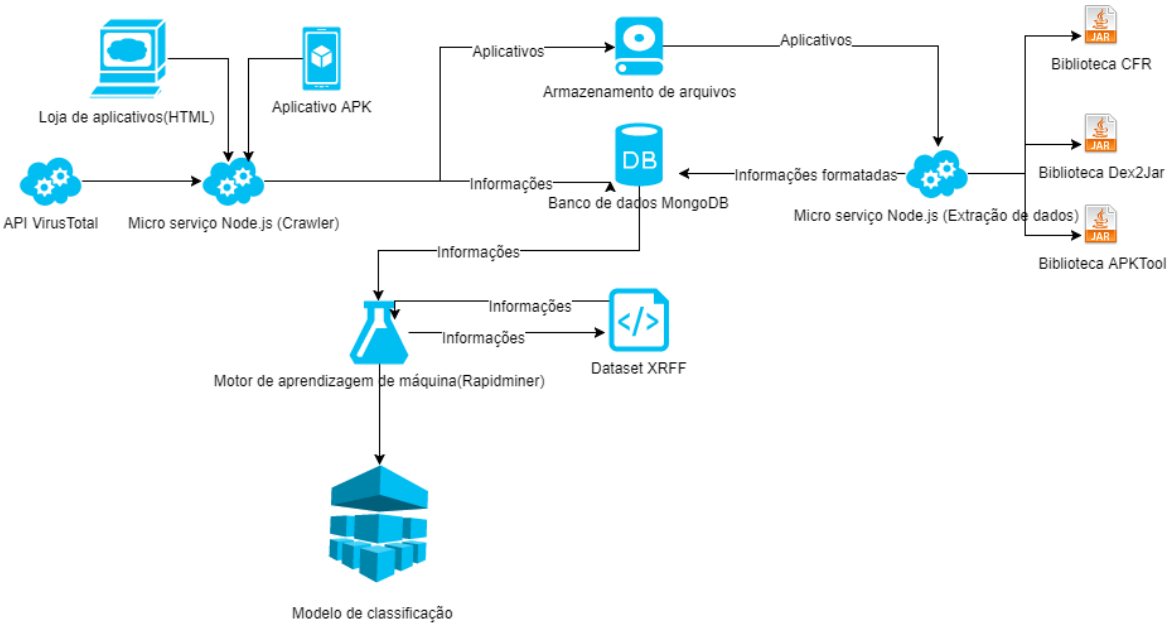
- Maior taxa de precisão;
- Menor taxa de falsos positivos;
- Menor taxa de falsos negativos;

Selecionar o melhor pode variar dependendo do objetivo da utilização do modelo, sendo a última atividade do processo, nesse caso, devemos priorizar menor taxa de falsos negativos (aplicativos infectados que estão sendo identificados como não infectados) e uma taxa de falsos positivos (aplicativos não infectados que são classificados como infectados) um pouco elevada é um bom sinal, pois isso pode significar que o modelo escolhido não está deixando de classificar corretamente aplicativos que de fato estão infectados e está classificando aplicativos não infectados com uma margem de erro, significando que pode estar classificando novos malwares ou apenas sendo uma abordagem com algum nível de precaução, se tratando de uma solução de segurança onde é preciso sempre prevenir a ameaça por menor que seja esse efeito colateral tende a ser benéfico.

## 4.2 VISÃO GERAL DA SOLUÇÃO

Nesta seção, será descrita a visão geral da solução utilizada para realizar a técnica proposta neste trabalho. A Figura 4.8, apresenta os componentes utilizados. Basicamente, existem três tipos de componentes: Serviços, armazenamento e classificação, que são utilizados para prover dados para o motor de aprendizagem de máquina que é utilizado para criar um modelo de classificação.

Figura 4.8 Visualização esquemática da solução.




Fonte: produção do próprio autor.

### 4.2.1 SERVIÇOS

O sistema é composto por dois micro serviços Node.js, utilizados localmente em uma máquina com a seguinte configuração: CPU com 4 núcleos, 8GB de memória RAM, SSD 120GB e conexão de internet de 15Mb; que são utilizados para realizar as etapas de extração de informações do aplicativos e metadados descritas na seção de visão geral, sendo eles: *Crawler/Scraper* e extração de dados.

O primeiro é responsável por coletar as informações da loja de aplicativos a partir da utilização de bibliotecas que imita o comportamento de um browser fazendo com que o servidor que hospeda a página, que é utilizada para capturar os dados, se apresente normalmente, permitindo a navegação automatizada pelo conteúdo da página. Ele também é responsável por realizar o download da aplicação APK e fazer requisições para a API do VirusTotal para identificar se a mesma está infectada, a partir de uma série de escaneamentos que a plataforma do VirusTotal aplica em diversos antivírus disponíveis no mercado.

O segundo micro serviço é responsável por três processos:

- Descompactar a aplicação APK em pastas;
- Extrair o manifesto com a biblioteca APKTool (TUMBLESON, Connor) para formato de texto legível (XML);

- Descompilar os componentes da aplicação, utilizando a biblioteca Dex2jar (PAN, Bob) para converter os arquivos DEX utilizados pelo sistema Android em arquivos JAVA CLASS, seguidos pela utilização da biblioteca CFR (BENFIELD, Lee) que converte arquivos JAVA CLASS em texto legível.

Uma vez realizado os processos descritos, todas as informações são persistidas no banco de dados.

### 4.2.2 ARMAZENAMENTO

Todos os aplicativos coletados das fontes descritas anteriormente foram armazenados de três formas:

- Informações referentes à aplicação a partir da loja ou internas ao aplicativo em um banco de dados MongoDB;
- Aplicativos em formato compactado original APK;
- Base de treinamento com aplicativos listados como amostras categorizados por atributos e armazenados em um arquivo em formato *eXtensible attribute-Relation File Format* (XRFF).

### 4.2.3 CLASSIFICAÇÃO

O motor de classificação escolhido foi a plataforma Rapidminer, pela facilidade em administrar processos de manipulação de dados, bem como a possibilidade de se conectar diretamente com a solução de banco de dados utilizada para coletar as informações, MongoDB, recuperando todas as informações formatadas dos aplicativos e realizando as etapas de pré-processamento, seleção de atributos e criação de modelo de classificação.

Descritos em dois subprocessos distintos, é possível definir a etapa de pré-processamento e seleção de atributos que consiste na geração de um arquivo resultado do tipo XRFF, que contém *samples* que formam uma base de dados com colunas que representam colunas do banco de dados antes da execução como no Quadro 4.7.

O formato de arquivo XRFF foi escolhido por ter compatibilidade com plataformas de aprendizagem de máquina e (Inteligência artificial) IA, tais como WEKA, Orange e Rapidminer. E possui colunas que representam atributos depois da

execução das etapas, como no Quadro 4.8, formando assim uma base de dados completa e preparada para um classificador utiliza-la como base de treinamento.

Quadro 4.5 Estrutura da base de dados.

| Infectado | Downloads | Tamanho | Descrição | ApiCalls | Intents | Permissões |
|---|---|---|---|---|---|---|
| Booleano | Inteiro | Texto | Texto | Texto | Texto | Texto |

Quadro 4.6 Estrutura da base de dados completa.

| Infectado | Downloads | Tamanho | Palavras (n atributos) | *ApiCalls* (n atributos) | *Intents* (n atributos) | Permissões (n atributos) |
|---|---|---|---|---|---|---|
| Booleano | Inteiro | Inteiro | Inteiro | Inteiro | Inteiro | Inteiro |

Com os atributos já selecionados e dispostos em um arquivo XRFF, outro subprocesso do Rapidminer é responsável por criar modelos de classificação com os algoritmos citados na etapa de classificação descrita na seção de visão geral, aplicando a técnica de validação cruzada para verificar a precisão e qualidade dos modelos a fim de se escolher manualmente o melhor para detectar aplicativos com *malware*, baseados nos critérios descritos na seção 4.1.

## 4.3 CONSIDERAÇÕES FINAIS

Este capítulo foi responsável por descrever a contribuição principal deste trabalho, que é uma técnica para detecção de ameaças de segurança em aplicativos *Android* a partir de análise estática e aprendizagem de máquina, modelada por meio do processo BPMN para facilitar a replicação das etapas por pesquisadores interessados em aprimorar a técnica ou utiliza-la para outros fins. Também foi detalhada uma visão geral da solução, onde os processos são executados por micro serviços, chamadas de API e utilização de um motor de aprendizagem de máquina popular no mercado.

O próximo capítulo se destina a ilustrar e avaliar a solução proposta, por meio da comparação da mesma com outros modelos.

# 5 AVALIAÇÃO

Este capítulo objetiva detalhar os resultados encontrados a partir da execução da solução proposta e também comparar estes resultados com outras soluções que utilizaram parâmetros similares de qualificação.

## 5.1 EXECUÇÃO DA TÉCNICA

A execução da abordagem proposta é descrita, nas próximas subseções, da forma que se segue: objetivos da avaliação, análise dos dados coletados, métricas de avaliação utilizadas e apresentação dos resultados encontrados.

### 5.1.1 OBJETIVOS

A abordagem detalhada no capítulo anterior ilustra uma técnica de detecção de *malwares* em aplicativos Android, porém para demonstrar a eficácia da mesma precisamos detalhar a natureza dos dados que foram utilizados, como foram abordados, categorizados e estruturados.

A partir dos dados coletados temos como aplicar todos os subprocessos que compõem a técnica e finalmente detalhar suas implicações com a base de dados montada.

### 5.1.2 BASE DE DADOS

Composta de um total de 2124 aplicativos, a base de dados coletada e formatada de três fontes distintas a serem detalhadas nas próximas subseções: 1Mobile, PlayDrone (VIENNOT, Nicolas; GARCIA, Edward; NIEH, Jason, 2014) e Contagio dump; foi analisada pela API do site VirusTotal para criar uma base de treinamento e testes consistente com os resultados de outros antivírus, resultando em 194 aplicativos infectados e 1930 não infectados.

Não foi possível identificar as quantidades exatas de aplicativos para cada fonte utilizada, pois vários scripts de coleta estavam sendo executados ao mesmo tempo, para diminuir o tempo de coleta.

As fontes descritas nas seções a seguir serviram para gerar registros similares ao mostrado anteriormente para cada aplicação coletada. Seguido pelo detalhamento da base de treinamento gerada a partir da base de dados.

5.1.2.1 1MOBILE

A loja de aplicativos 1Mobile fornece aplicativos grátis e possui uma estrutura de aplicação Web relativamente fácil para detectar os pontos chave para criar um *crawler/scraper* para coletar todas as informações possíveis sobre cada aplicativo. Sendo assim foram coletadas informações dos seguintes campos, além da aplicação em si:

- Nome;
- Descrição;
- Categoria;
- Tamanho;
- Data de atualização;
- Número de downloads;
- Versão do Android requerida;

*5.1.2.2 PLAY STORE*

O projeto *Playdrone* (VIENNOT, Nicolas; GARCIA, Edward; NIEH, Jason, 2014) foi criado para agregar todas as aplicações da loja da Google *Play Store* no ano de 2014 em um determinado dia. Como os dados estão estruturados com todas as informações previamente citadas na coleta de aplicativos da loja 1Mobile, essa base de dados pôde ser usada para complementar a base de dados previamente citada.

*5.1.2.3 CONTAGIO DUMP*

Grande parte dos aplicativos infectados tem origem a partir de extração manual do site *contagio dump*, que é conhecido por ter vírus, *malwares* e afins compartilhados pelos usuários, fazendo com que seja uma fonte atualizada desse tipo de material. No entanto não há informações da loja que o aplicativo estava antes de ser compartilhado no site.

**5.1.3 BASE DE TREINAMENTO**

A partir das técnicas de extração e seleção de atributos, foram gerados os seguintes quadros que se referem aos atributos utilizados nos modelos de

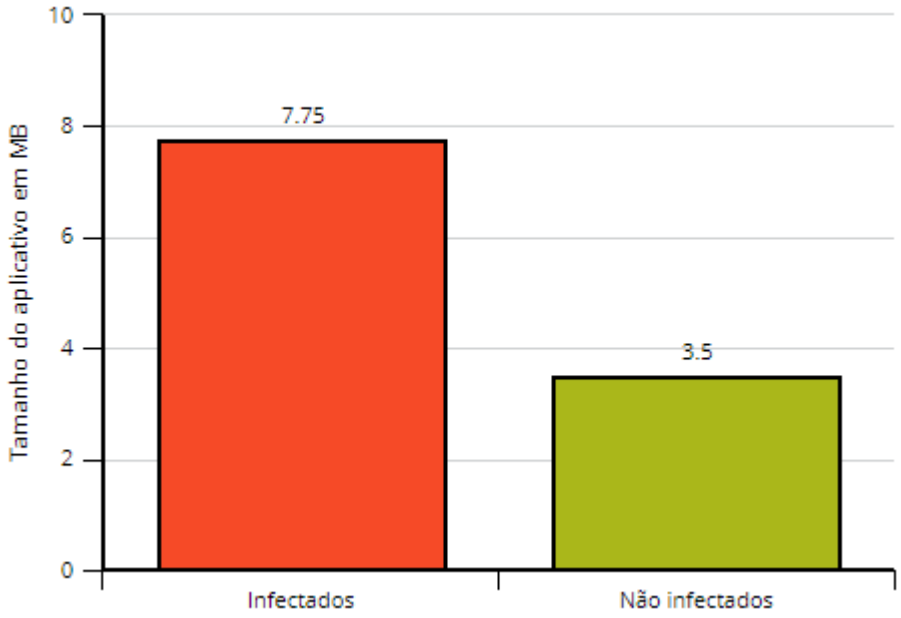
classificação. Todos os atributos estão nomeados e foram extraídos com base na língua inglesa, tendo em vista que as fontes estão em inglês.

Quadro 5.1 Atributos relacionados ao texto de descrição da loja e outras informações

| Atributos relacionados às lojas | Tipo |
|---|---|
| Tamanho da aplicação em MB | Inteiro (normalizado) |
| *abstract* | Booleano |
| *bold* | Booleano |
| *buni* | Booleano |
| *calibration* | Booleano |
| *chill* | Booleano |
| *colourful* | Booleano |
| *cooper* | Booleano |
| *deactivated* | Booleano |
| *desctop* | Booleano |
| *expand* | Booleano |
| *fanatic* | Booleano |
| *grade* | Booleano |
| *graphical* | Booleano |
| *harlem* | Booleano |
| *icicle* | Booleano |
| *iranian* | Booleano |
| *jigsaw* | Booleano |
| *obadiah* | Booleano |
| *paired* | Booleano |
| *pakistan* | Booleano |
| *perpetual* | Booleano |
| *personalization* | Booleano |
| *proximity* | Booleano |
| *sidet* | Booleano |
| *speedy* | Booleano |
| *telangana* | Booleano |
| *telephony* | Booleano |

No quadro 5.1 temos a lista de atributos relacionados às lojas de aplicativos, contendo o atributo de tamanho da aplicação que acabou sendo útil, pois foi possível notar que os aplicativos de maior tamanho têm maiores chances de estarem infectados, como ilustrado na Figura 5.1.

Figura 5.1 Relação entre aplicativos infectados e tamanho médio de arquivos da aplicação.




Fonte: Produção do próprio autor.

Apesar das palavras da descrição da lista serem um pouco aleatórias, é possível relacionar palavras como *telephony* e *paired* a aplicações que utilizam características de comunicação, podendo até ligar essa informação a malwares que utilizam vulnerabilidades de acesso à chamadas, envio e recebimento de mensagens no celular.

Outra observação importante se dá ao fato de que palavras podem estar se referindo à customização do sistema operacional, tais como: *calibration, colourful, graphical, expand, personalization, deactivated*. Neste caso, pode-se assim justificar o uso de permissões mais invasivas sem que o usuário perceba as reais intenções do desenvolvedor da aplicação ou de um terceiro que modificou a aplicação e está se passando por um desenvolvedor.

No quadro 5.2 temos atributos de chamadas de aplicação, referentes ao código fonte extraído dos aplicativos.

Quadro 5.2 Atributos relacionados a chamadas de aplicação no código (todos booleanos).

| API Calls |
|---|
| *getDeviceId* |
| *getInputStream* |
| *getInstalledPackages* |
| *getMessageBody* |
| *getRunningAppProcesses* |
| *getRunningServices* |
| *getSimSerialNumber* |
| *installPackage* |
| *saveBookmark* |
| *sendMultipartTextMessage* |
| *sendTextMessage* |
| *silenceRinger* |

Após análise do quadro 5.2, pode-se inferir que os atributos mais utilizados são de coleta de informações do usuário (*getDeviceId, getInputStream, getInstalledPackages, getMessageBody, getRunningAppProcesses, getRunningServices, getSimSeriaNumber*), que é uma área de ataque comum dos malwares em aparelhos móveis. Assim como a manipulação de software sem o consentimento do usuário, com as chamadas de aplicação *installPacakge, saveBookmark, sendMultipartTextMessage, sendTextMessage, silenceRinger*.

Posteriormente, no Quadro 5.3, atributos aliados às permissões de sistema descritas no manifesto das aplicações coletadas.

Quadro 5.3 Atributos relacionados a permissões (todos booleanos).

| Permissões |
|---|
| *android.permission.ACCESS_LOCATION_EXTRA_COMMANDS* |
| *android.permission.ACCESS_WIFI_STATE* |
| *android.permission.CALL_PRIVILEGED* |
| *android.permission.CHANGE_NETWORK_STATE* |
| *android.permission.CHANGE_WIFI_MULTICAST_STATE* |
| *android.permission.CLEAR_APP_CACHE* |
| *android.permission.DELETE_PACKAGES* |
| *android.permission.DISABLE_KEYGUARD* |
| *android.permission.DUMP* |
| *android.permission.FLASHLIGHT* |
| *android.permission.GET_PACKAGE_SIZE* |
| *android.permission.INJECT_EVENTS* |
| *android.permission.INSTALL_PACKAGES* |
| *android.permission.INTERACT_ACROSS_USERS* |
| *android.permission.INTERACT_ACROSS_USERS_FULL* |
| *android.permission.INTERNET* |
| *android.permission.LOCATION_HARDWARE* |
| *android.permission.MODIFY_PHONE_STATE* |
| *android.permission.MOUNT_FORMAT_FILESYSTEMS* |
| *android.permission.MOUNT_UNMOUNT_FILESYSTEMS* |
| *android.permission.READ_PROFILE* |
| *android.permission.READ_SYNC_SETTINGS* |
| *android.permission.REORDER_TASKS* |
| *android.permission.RESTART_PACKAGES* |
| *android.permission.SET_DEBUG_APP* |
| *android.permission.SET_WALLPAPER* |
| *android.permission.WAKE_LOCK* |
| *android.permission.WRITE_APN_SETTINGS* |
| *android.permission.WRITE_SYNC_SETTINGS* |
| *android.provider.Telephony.SMS_RECEIVED* |
| *android.software.leanback* |
| *com.android.alarm.permission.SET_ALARM* |
| *com.android.launcher.permission.INSTALL_SHORTCUT* |

As permissões de controle de dispositivo, instalação de novos pacotes e mudanças no sistema são comuns, pois acabam gerando vulnerabilidades de sistema que podem ser utilizadas por ameaças como *malwares*. E na lista acima pode-se notar que são a maioria, com exceção do acesso à lanterna do smartphone que talvez seja uma permissão desnecessária de ter sido selecionada, podendo estar ligada à aplicativos que não estão infectados.

Por fim o quadro 5.4 tem as informações de *intents* e *features* de hardware do aparelho com o sistema Android contidos no manifesto da aplicação.

Quadro 5.4 Atributos relacionados a *intents* e *features(todos booleanos).*

| *Intents e features* |
|---|
| *android.hardware.microphone* |
| *android.hardware.nfc* |
| *android.hardware.screen.landscape* |
| *android.hardware.screen.portrait* |
| *android.hardware.bluetooth_le* |
| *android.hardware.telephony* |
| *android.hardware.touchscreen* |
| *android.hardware.touchscreen.multitouch* |
| *android.hardware.usb.host* |
| *android.intent.action.ACTION_POWER_CONNECTED* |
| *android.intent.action.ACTION_POWER_DISCONNECTED* |
| *android.intent.action.BATTERY_LOW* |
| *android.intent.action.BOOT_COMPLETED* |
| *android.intent.action.DATE_CHANGED* |
| *android.intent.action.DOWNLOAD_COMPLETE* |
| *android.intent.action.MAIN* |
| *android.intent.action.MUSIC_PLAYER* |
| *android.intent.action.PICK* |
| *android.intent.action.SEARCH* |
| *android.intent.action.SEND* |
| *android.intent.action.SET_WALLPAPER* |
| *android.intent.action.TIME_TICK* |
| *android.intent.action.USER_PRESENT* |
| *android.intent.action.VIEW* |
| *android.media.AUDIO_BECOMING_NOISY* |

No Quadro 5.4 é possível notar que o acesso a microfone, (Universal Serial Bus) USB, *bluetooth*, telefonia e ações físicas do usuário, tais como toque na tela, usuário presente e status de conexões podem ser tratados como possíveis formas de roubo de informações do usuário por meio de vulnerabilidades de segurança que são exploradas por o aplicativo permitir esse tipo de interação e ter sido aceita pelo usuário ao dar acesso de permissão a tais configurações e indicadores de eventos de sistema. Desta forma, tais atributos se mostram relevantes na detecção de aplicativos infectados.

### 5.1.4 TÉCNICAS DE AVALIAÇÃO

Foram utilizadas três técnicas de avaliação individual de classificação no trabalho, sendo elas: precisão de classificação e matrizes de confusão que são compostas de falsos positivos e falsos negativos.

#### 5.1.4.1 PRECISÃO

Os classificadores testados foram avaliados em um processo de validação cruzada com utilização da técnica *k-fold*, com k igual a 10, onde a base de treinamento é dividida em 10 partes para realizar a técnica de validação cruzada, bem como amostras separadas de forma balanceada com o mesmo número de amostras infectadas e não infectadas em cada parte.

O Quadro 5.5 demonstra que o classificador com melhor precisão é o Random Forest. Contudo, todos estão em níveis aceitáveis de classificação acima de 80%, indicando que os resultados possivelmente estão dentro de uma margem de segurança aceitável e não estão sofrendo de *overfitting,* ou seja, não estão apenas encontrando padrões de classificação relacionados estritamente à base de treinamento utilizada, podendo ser utilizados com outros dados e provavelmente obter resultados semelhantes.

Quadro 5.5 Precisão dos classificadores testados

| Classificador | Precisão |
|---|---|
| *Random Forest* | 88.42% |
| Árvore de decisão | 86.14% |
| *Deep Learning* | 84.35% |
| SVM | 83.66% |

#### 5.1.4.2 MATRIZES DE CONFUSÃO

No quadro 5.6 temos as informações referentes à matriz de confusão de cada classificador utilizado nesse trabalho, separados em falsos positivos e falsos negativos. Lembrando que 1000 é o número total de amostras, 500 em cada classe.

Quadro 5.6 Falsos positivos e falsos negativos

| Classificador | Falsos positivos | Falsos negativos |
|---|---|---|
| *Random Forest* | 184 | 42 |
| Árvore de decisão | 29 | 78 |
| *Deep Learning* | 60 | 90 |
| SVM | 92 | 80 |

## 5.2 AVALIAÇÃO COMPARATIVA

A abordagem definida para comparar o trabalho com outros de natureza semelhante foi a de comparação de resultados do melhor classificador/algoritmo de cada solução.

Sendo assim, os resultados de alguns trabalhos relacionados que também objetivam detectar ameaças de segurança são próximos ao da abordagem proposta, sendo eles: taxa de precisão(92% à 100%), falsos positivos(0% à 10%), falsos negativos(0% à 9%) e tamanho da base de dados por quantidade de aplicativos testados(230 à 1600).

Pode-se observar, por meio dos dados apresentados anteriormente, que este trabalho apresentou um resultado de precisão semelhante aos já existentes na literatura, onde ele leva certa desvantagem em alguns quesitos, mas que as distâncias de desempenho para os já existentes mostram que a técnica proposta continua sendo uma abordagem válida, e que pode ser usada em conjunto com as outras para se obter resultados ainda melhores. Deve se levar em consideração que a abordagem descrita nesse trabalho se propõe a utilizar outras fontes de dados para classificar os aplicativos em infectados ou não, dessa forma se estivéssemos comparando a mesma base de dados os resultados provavelmente seriam coerentes para uma comparação quantitativa.

Temos um comparativo qualitativo, representando a proximidade dos resultados, entre este trabalho e os citados, onde vemos que falsos positivos

aparecem em taxas relativamente altas, mas acabam não sendo motivo para descreditar nenhuma abordagem, pois o oposto, que seria a detecção de falsos negativos em detrimento dos falsos positivos significaria que a técnica de detecção não está identificando ameaças reais, o que não é aceitável para abordagens de segurança.

## 5.3 CONSIDERAÇÕES FINAIS

A avaliação da técnica proposta foi responsável por classificar a mesma dentre suas concorrentes e abordagens similares, mostrando que ela apresenta resultados semelhantes de forma qualitativa a técnicas já existentes mesmo não utilizando dados do mesmo tipo para a aplicação da mesma. Esse tipo de comparação é útil para ilustrar a eficácia da proposta, e em especial mostrando que ela pode ser usada em conjunto com as outras técnicas para realizar a complexa tarefa de classificar *malwares*. A seguir veremos as conclusões deste trabalho e possíveis melhorias que podem ser feitas nesse trabalho na continuidade futura deste trabalho.

# 6 CONCLUSÕES E TRABALHOS FUTUROS

## 6.1 CONCLUSÕES

O objetivo principal do trabalho foi propor uma estratégia de utilização de meta informações referentes à meios externos ao aplicativo aliados à análise do código da aplicação e aprendizado de máquina com a finalidade de detecção de *malwares*. Foi mostrado que a técnica proposta é interessante, pois os resultados encontrados foram semelhantes aos trabalhos relacionados.

Os atributos obtidos a partir dos metadados aparecem em maior quantidade perante aos obtidos por meio do código fonte da aplicação e ainda assim o modelo de classificação que utiliza a base de treinamento formada por eles possui boa precisão na detecção de ameaças. É possível dizer que a combinação das duas técnicas aplicadas tem benefícios, ou pelo menos não afetam negativamente a detecção de ameaças.

A abordagem proposta neste documento acaba tornando o processo de verificação de ameaças aliado à qualidade do aplicativo em si, apesar de acabar causando algum nível de *overfitting*, pela quantidade de atributos que puderam ser utilizados em relação à quantidade de amostradas utilizadas para realizar a classificação.

## 6.2 TRABALHOS FUTUROS

Diversas possibilidades de trabalhos são vislumbradas no contexto deste trabalho, e elas estão descritas a seguir.

- **Aplicar a estratégia em bases de dados de maior escala.** Provavelmente um aumento na quantidade de amostras iria resultar em taxas de detecção de *malwares* mais próximas de taxas utilizadas no mercado, tendo em vista que poderiam ser utilizados mais atributos de diversas fontes;
- **Uso de técnicas de partição binária da aplicação compilada**. Utilizar informações do aplicativo sem aplicar o processo de descompilaçao poderia agregar valor à qualidade das informações obtidas por meio do arquivo da aplicação, pois a ofuscação do código, que consiste em modificar o código

fonte a ponto de não ser legível, porém ainda funcional; não iria ser um fator prejudicial;

- **Verificar a língua das palavras na descrição dos aplicativos**. Os termos utilizados na técnica proposta são na língua inglesa, portanto adicionar compatibilidade com outras línguas iria possibilitar a utilização de mais meta informação sobre o aplicativo;
- **Separar aplicativos infectados em classes de ameaças**. Tornar evidente qual tipo de ameaça está sendo detectada é uma forma de identificar perfis específicos e aumentar a capacidade de utilização da abordagem;
- **Utilizar fontes de análise inicial de base de dados diversificadas**. Utilizar dados de outras lojas de aplicativos sobre o mesmo arquivo da aplicação pode trazer diferenças que identifique uma potencial modificação nociva da aplicação.
- **Utilizar outras técnicas de aprendizagem de máquina**. Além de melhorias nas técnicas de extração e seleção de atributos, outros tipos de técnicas de classificação podem ser utilizados para melhorar taxas de detecção e reduzir falsos positivos, sendo elas composições de classificadores e redes neurais de alta complexidade.

# 7 REFERÊNCIAS

1Mobile. Disponível em http://www.1mobile.com/ . Acesso em: setembro de 2017.

AAFER, Yousra; DU, Wenliang; YIN, Heng. DroidAPIMiner: Mining API-level features for robust malware detection in android. In: **International Conference on Security and Privacy in Communication Systems**. Springer International Publishing, 2013. p. 86-103.

ALAM, Mohammed S.; VUONG, Son T. Random forest classification for detecting android malware. In: **Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing**. IEEE, 2013. p. 663-669.ZHAO, Min et al.

APVRILLE, Ludovic; APVRILLE, Axelle. Identifying Unknown Android Malware with Feature Extractions and Classification Techniques. In: **Trustcom/BigDataSE/ISPA, 2015 IEEE**. IEEE, 2015. p. 182-189.

BENFIELD, Lee. **CFR**. Disponível em http://www.benf.org/other/cfr/ . Acesso em setembro de 2017.

BRAGA, Alexandre Melo; DO NASCIMENTO, Erick Nogueira; RODRIGUES, Lucas. Introdução à Segurança de Dispositivos Móveis Modernos–Um Estudo de Caso em Android. **Simpósio em Segurança da Informação e de Sistemas Computacionais. Sociedade Brasileira de Computação–SBC**, v. 12, 2012.

CRUSSELL, Jonathan; GIBLER, Clint; CHEN, Hao. Attack of the clones: Detecting cloned applications on android markets. In: **European Symposium on Research in Computer Security**. Springer Berlin Heidelberg, 2012. p. 37-54.

FARUKI, Parvez et al. AndroSimilar: robust statistical feature signature for Android malware detection. In: **Proceedings of the 6th International Conference on Security of Information and Networks**. ACM, 2013. p. 152-159.

FRATANTONIO, Yanick et al. CLAPP: characterizing loops in Android applications. In: **Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering**. ACM, 2015. p. 687-697.

HOARE, Leslie. **JD-Core**. Disponível em http://jd.benow.ca/ .Acesso em: setembro de 2017.

KANG, Hyunjae et al. Detecting and classifying android malware using static analysis along with creator information. **International Journal of Distributed Sensor Networks**, v. 2015, p. 7, 2015.

LI, Qi; LI, Xiaoyu. Android malware detection based on static analysis of characteristic tree. In: **Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2015 International Conference on**. IEEE, 2015. p. 84-91.

**MathWorks**. Disponível em https://www.mathworks.com/help/stats/zscore.html?s_tid=gn_loc_drop. Acesso em: agosto de 2017.

MUÑOZ, Alfonso et al. Android malware detection from Google Play meta-data: selection of important features. In: **Communications and Network Security (CNS), 2015 IEEE Conference on**. IEEE, 2015. p. 701-702.

PAN, Bob. **Dex2Jar**. Disponível em https://github.com/pxb1988/dex2jar .Acesso em: Setembro de 2017.

PEHLIVAN, Uğur et al. The analysis of feature selection methods and classification algorithms in permission based Android malware detection. In: **Computational Intelligence in Cyber Security (CICS), 2014 IEEE Symposium on**. IEEE, 2014. p. 1-8.

PEIRAVIAN, Naser; ZHU, Xingquan. Machine learning for android malware detection using permission and api calls. In: **2013 IEEE 25th International Conference on Tools with Artificial Intelligence**. IEEE, 2013. p. 300-305.

RIBEIRO, Athos Coimbra. Análise Estática de Código-Fonte com Foco em Segurança: Metodologia Para Avaliação de Ferramentas, 2015.

SANZ, Borja et al. MAMA: manifest analysis for malware detection in android. **Cybernetics and Systems**, v. 44, n. 6-7, p. 469-488, 2013.

SATO, Ryo; CHIBA, Daiki; GOTO, Shigeki. Detecting Android malware by analyzing manifest files. **Proceedings of the Asia-Pacific Advanced Network**, v. 36, p. 23-31, 2013.

SHABTAI, Asaf; FLEDEL, Yuval; ELOVICI, Yuval. Automated static code analysis for classifying android applications using machine learning. In: **Computational Intelligence and Security (CIS), 2010 International Conference on**. IEEE, 2010. p. 329-333.

Statista. **Android OS of global smartphone market share**. https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. Acesso em Agosto de 2017.

Statista. **TECH Android Poised to Knock Windows off Internet Perch**. https://www.statista.com/chart/8449/android-poised-to-knock-windows-off-internet-perch/. Acesso em Agosto de 2017.

TUMBLESON, Connor. **ApkTool**. Disponível em http://ibotpeaches.github.io/Apktool/. Acesso em: setembro de 2017.

VIENNOT, Nicolas; GARCIA, Edward; NIEH, Jason. A measurement study of google play. In: **ACM SIGMETRICS Performance Evaluation Review**. ACM, 2014. p. 221-233.

**VirusTotal**. Disponível em https://www.virustotal.com/pt/. Acesso em: setembro de 2017.

WEI, Te-En et al. Android malware detection via a latent network behavior analysis. In: **2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications**. IEEE, 2012. p. 1251-1258.

WONG, Michelle Y.; LIE, David. IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware. In: **Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)**. 2016.

WU, Dong-Jie et al. Droidmat: Android malware detection through manifest and api calls tracing. In: **Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on**. IEEE, 2012. p. 62-69.

YERIMA, Suleiman Y. et al. A new android malware detection approach using bayesian classification. In: **Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on**. IEEE, 2013. p. 121-128.

YERIMA, Suleiman Y.; SEZER, Sakir; MUTTIK, Igor. Android malware detection using parallel machine learning classifiers. In: **2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies**. IEEE, 2014. p. 37-42.

YUAN, Zhenlong; LU, Yongqiang; XUE, Yibo. Droiddetector: android malware characterization and detection using deep learning. **Tsinghua Science and Technology**, v. 21, n. 1, p. 114-123, 2016.

ZHAO, Min; GE, Fangbin; ZHANG, Tao; YUAN, Zhijian; AntiMalDroid: an efficient SVM-based malware detection framework for Android. In: **International Conference on Information Computing and Applications**. Springer Berlin Heidelberg, 2011. p. 158-166.

ZHENG, Min; SUN, Mingshen; LUI, John CS. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In: **Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on**. IEEE, 2013. p. 163-171.

ZHOU, Wu et al. Detecting repackaged smartphone applications in third-party android marketplaces. In: **Proceedings of the second ACM conference on Data and Application Security and Privacy**. ACM, 2012. p. 317-326.

ZHOU, Yajin et al. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In: **NDSS**. 2012. p. 50-52.

ZHIOUA, Zeineb; SHORT, Stuart; ROUDIER, Yves. Static Code Analysis for Software Security Verification: Problems and Approaches. In: **Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International**. IEEE, 2014. p. 102-109.