DETECÇÃO DE ANDROID BOTNET BASEADA NA RELEVÂNCIA DE PERMISSÕES E FILTROS DE INTENÇÃO

IGOR FELIPE SODRÉ RIBEIRO CARNEIRO

DETECÇÃO DE ANDROID BOTNET BASEADA NA RELEVÂNCIA DE PERMISSÕES E FILTROS DE INTENÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Instituto de Ciências Exatas da Universidade Federal do Amazonas como requisito parcial para a obtenção do grau de Mestre em Informática.

Orientador: Eduardo James Pereira Souto, DSc
Coorientador: Thiago de Souza Rocha, DSc

Manaus

Agosto de 2022

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Carneiro, Igor Felipe Sodré Ribeiro
C289d Detecção de android botnet baseada na relevância de permissões e filtros de intenção / Igor Felipe Sodré Ribeiro Carneiro . 2022
63 f.: il. color; 31 cm.

Orientador: Eduardo James Pereira Souto
Coorientador: Thiago de Souza Rocha
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Android. 2. Botnet. 3. Android botnets. 4. Dispositivos móveis. 5. Classificação. I. Souto, Eduardo James Pereira. II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Detecção de Android Botnet Baseada na Relevância de Permissões e Filtros de Intenção"

IGOR FELIPE SODRÉ RIBEIRO CARNEIRO

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Eduardo James Pereira Souto - PRESIDENTE

Prof. Eduardo Luzeiro Feitosa- MEMBRO INTERNO

Prof.Gilbert Breves Martins- MEMBRO EXTERNO

Manaus, 09 de Agosto de 2022

Resumo

O grande número de dispositivos Android e a disponibilidade de dados sensíveis tornaram os *smartphones* um novo ambiente para propagação de atividades maliciosas. Como os *smartphones* tendem a ficar online por longos períodos, eles fornecem uma plataforma ideal para operar *botnets*, também conhecidas como Android *botnets*. Por essa razão, pesquisas recentes têm direcionado seus esforços em soluções de detecção de Android *botnets* baseadas em informações presentes nos aplicativos. Entretanto, a falta de entendimento do comportamento e especificidades dos *malwares* presentes em *botnets* para dispositivos móveis dificultam o projeto de soluções para mitigar esse problema. Para tornar os sistemas de detecção de *botnets* mais eficientes, discriminar as características que descrevem aplicativos benignos e maliciosos é uma questão crítica e fundamental para o desenvolvimento de contramedidas. Neste contexto, este trabalho descreve um método de detecção de Android *botnet* baseado em dados extraídos de aplicativos Android utilizando quantificadores de recuperação da informação para definir as características mais relevantes e, como resultado, proporcionar maior eficácia de detecção de Android *botnet* por meio de algoritmos de aprendizado de máquina. O método proposto reduz a dimensionalidade do espaço de características usando uma medida de ponderação baseada no TF-IDF (Term Frequency-Inverse Document Frequency) para identificar as características mais relevantes em cada amostra por meio dos conjuntos de permissões solicitadas ao usuário e das ações executadas pelos componentes da aplicação. Experimentos realizados com 2.997 amostras reais de aplicativos (benignos e maliciosos) mostram que o método proposto melhora, em todos os cenários avaliados, a eficácia de modelos de aprendizagem no processo de classificação de Android *botnets*.

Palavras-chave: Android, Botnet, Android Botnets, Dispositivos Móveis, Classificação, Malwares, Detecção de Malware.

Abstract

A large number of Android devices and the availability of sensitive data have made smartphones a new environment for spreading malicious activities. As smartphones remain online for long periods, they provide an ideal platform for operating botnets, also known as Android botnets. For this reason, recent research has focused on solutions for Android botnet detection based on information from applications. However, the lack of understanding of the behavior and specifics of malware in botnets for mobile devices makes it difficult to design solutions to mitigate this problem. To make botnet detection systems more efficient, discriminating the characteristics that describe benign and malicious applications is a critical and fundamental issue for developing countermeasures. In this context, this work describes an Android botnet detection method based on data extracted from Android applications using information retrieval quantifiers to define the most relevant characteristics and, as a result, provide greater efficiency of Android botnet detection through machine learning algorithms. The proposed method reduces the feature space dimensionality using a weighting measure based on the TF-IDF (Term Frequency-Inverse Document Frequency) to identify the most relevant features in samples through requested permissions and the actions performed by the application components. Experiments performed with 2,997 real world samples of applications (benign and malicious) show that the proposed method improves, in all evaluated scenarios, the effectiveness of learning models in the classification process of Android botnets.

Palavras-chave: Android, Botnet, Android Botnets, Smartphones, Classification, Malwares, Malware Detection.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Os *smartphones* passaram a ser um importante recurso em nosso cotidiano, visto que possibilitam o acesso a uma variedade de serviços ubíquos como transações financeiras, compras online, redes sociais, *streaming* de música, envio de mensagens multimídia, entre outros. Essa infinidade de recursos levou a uma difusão generalizada destes dispositivos que, como resultado, passaram a ser um alvo preferencial de atacantes (Zulkefli & Mahinderjit Singh, 2020). Diariamente, diversos dispositivos de usuários são infectados por meio da instalação de aplicações maliciosas para, por exemplo, obter acesso não autorizado a dados confidenciais dos usuários, propagar envio de mensagens não solicitadas (*spam*) ou mesmo causar a indisponibilidade de serviços por meio de *botnets* móveis (Karim et al., 2015; Kirubavathi & Anitha, 2018; Koyuncu & Pusatli, 2019; Hojjatinia et al., 2020).

Botnets ainda são consideradas uma das maiores ameaças digitais e durante um tempo seus principais alvos foram os computadores pessoais (Farina et al., 2016). Entretanto, a disponibilidade de informações privadas e o acesso a Internet móvel tornou os *smartphones* um novo ambiente de propagação de atividades maliciosas (Talal et al., 2019; González et al., 2021). Neste sentido, o sistema operacional Android[1], um dos mais populares no mundo, tornou-se alvo preferencial dos atacantes, dando origem ao que conhecemos como Android *botnet* (Hijawi et al., 2017; Kothari & Joshi, 2020)

Semelhante a uma *botnet* tradicional baseada em computadores *desktop*, uma Android *botnet* refere-se a um grupo de *smartphones* infectados (*bots*) por códigos maliciosos e controlados remotamente pelo atacante (*botmaster*) por meio de servidores de comando e controle (C&C - *Command & Control*) (Hojjatinia et al., 2020; Derakhshan & Ashrafnejad, 2020). Este comportamento de comunicação é encontrado em vários *malwares* para Android, como Geinimi, PJapps, NickySpy, TigerBot e Wroba. Tais

[1] https://www.android.com/

malwares tem sido encontrados tanto em lojas de aplicativos de terceiros como na loja oficial do Android (Hojjatinia et al., 2020).

Devido a falta de eficácia dos métodos de segurança conduzidos pelas diferentes lojas de aplicativos, os atacantes são capazes de propagar *malwares* para Android de forma rápida e discreta, ao se passar como um aplicativos legítimos (Talal et al., 2019). Para reagir de maneira mais efetiva a grande quantidade de *malwares* existentes, analistas de segurança procuram encontrar padrões comportamentais exclusivos para cada amostras de *malware*, visto que os desenvolvedores maliciosos criam variantes pela reutilização de códigos principais (Jang et al., 2014). As soluções existentes geralmente avaliam o conjunto de *malwares* com base no uso de permissões (Hojjatinia et al., 2020; Şahın et al., 2018), utilização de componentes de *software* e *hardware* (Kirubavathi & Anitha, 2018; Alqatawna et al., 2021) ou chamadas de API (Yusof et al., 2017; Yerima et al., 2021). Entretanto, a falta de entendimento do comportamento e especificidades dos *malwares* presentes em *botnets* para dispositivos móveis, dificulta o projeto de soluções para mitigar esse problema.

A comunicação entre componentes é um dos recursos primordiais do Android para acessar diferentes serviços da estrutura do sistema operacional e, portanto, pode ser usada para obter informações a respeito do comportamento das aplicações, sejam elas benignas ou maliciosas. Desta forma, além do uso de permissões, o filtro de intenção pode ser utilizado para caracterizar e auxiliar no processo de detecção de *Android botnet*. Um filtro de intenção é um objeto de mensagem que pode ser usado para solicitar uma ação de outro componente do aplicativo. Neste sentido, este trabalho visa fornecer um estudo baseado em análise exploratória e um método para detectar Android *botnets* por meio da utilização de diferentes conjuntos de características relevantes.

1.1 Problema

Neste trabalho, estudamos o problema de detecção de *botnets* móveis na plataforma Android. Dado um aplicativo, nosso objetivo é determinar se o mesmo é benigno ou malicioso (Android *botnet*).

Trabalhos anteriores exploram padrões de detecção baseados em vetores de características como permissões (Şahın et al., 2018; Hojjatinia et al., 2020) e recursos de *hardware* (Kirubavathi & Anitha, 2018) para identificar a presença e determinar a importância das caraterísticas. Entretanto, além de tais abordagens levar a uma matriz esparsa de alta dimensão que pode ser restringida por limites de memória, a quantidade de permissões idênticas ou de relevância semelhante nos conjuntos de amostras

benignas e maliciosas podem induzir ao erro no processo de detecção.

Para resolver essas limitações, propomos uma abordagem de classificação que utiliza medida de ponderação padrão utilizada na área de recuperação de informação, para definir os pesos de cada característica de modo que posteriormente a seleção das características mais relevantes seja aplicada, visando atender a redução da dimensionalidade. Além disso, nossa abordagem também propõe um conjunto de características baseado em permissões e filtros de intenção, que são recursos presentes em aplicativos benignos e maliciosos, e que auxiliam no melhor entendimento sobre os principais comportamentos das amostras analisadas.

1.2 Objetivos

Para lidar com o problema mencionado acima, esta pesquisa tem como objetivo desenvolver um método de detecção de Android *botnet* baseado em dados extraídos de aplicativos Android utilizando quantificadores de recuperação da informação para definir as características mais relevantes, assim proporcionar maior eficácia de detecção de Android *botnet* por meio de algoritmos de aprendizado de máquina;

Para alcançar o objetivo geral dessa pesquisa, um conjunto de objetivos específicos devem ser alcançados, dentre os quais destacam-se:

1. Aplicar um estudo de análise exploratória para entender padrões entre características distintas que representem amostras benignas e de Android *botnet*.

2. Definir um mecanismo de seleção de características dos dados coletados dos aplicativos que servirá de entrada para a criação do modelo de detecção de Android *botnet*.

3. Definir um mecanismo de que combine as características mais relevantes para detectar Android *botnet*.

1.3 Organização da Dissertação

O restante desta dissertação está organizado como segue:

O Capítulo 2 descreve os principais conceitos utilizados neste trabalho, fornecendo informações necessárias para o entendimento da pesquisa realizada.

O Capítulo 3 apresenta os trabalhos relacionados, divididos em dois grupos: detecção de *malwares* e detecção de *botnet* móvel. Além disso, neste capítulo são apresentados os trabalhos utilizados para efeitos de comparação com esta pesquisa.

O Capítulo 4 apresenta a visão geral do método proposto, dividido em quatro etapas: A primeira etapa descreve o processo de descompilação, onde todos os aplicativos são submetidos para que os artefatos de estudo sejam obtidos. A segunda etapa apresenta o processo de extração de características baseada na lista de permissões e filtros de intenção obtidas por meio de arquivos de configuração de cada aplicativo. A terceira etapa discorre sobre o processo de seleção de características, assim como apresenta a utilização da medida de ponderação para auxiliar na identificação das características mais relevantes. Por fim, a última etapa apresenta o processo de detecção de Android *botnet*, por meio de algoritmos de aprendizagem de máquina.

O Capítulo 5 apresenta os experimentos e resultados obtidos por meio do estudo realizado para detecção de Android *botnet*, assim como a comparação com outros trabalhos da literatura. Além disso, resultados referentes ao estudo de classificação de famílias de Android *botnet* também são apresentados.

Por fim, o Capítulo 6 apresenta as considerações finais do trabalho e sugestões de trabalhos futuros para que outros trabalhos possam evoluir o método proposto com adição de melhorias.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentados os conceitos prévios necessários para o entendimento e desenvolvimento do trabalho. A Seção 2.1, apresenta os principais fundamentos sobre o sistema operacional Android, sua arquitetura, componentes e as principais características de uma aplicação. A Seção 2.2 apresenta os conceitos sobre *botnets* e os seus aspectos fundamentais. A Seção 2.3 apresenta as principais abordagens para análise de aplicações maliciosas. A Seção 2.4 aborda as técnicas de aprendizagem de máquina empregadas pela literatura assim como discorre sobre técnicas de avaliação de modelos. Por fim, na Seção 2.5 encontram-se as considerações finais deste capítulo.

2.1 Android

O Android foi lançado pelo Google em 2008 como sistema operacional móvel, oferecendo suporte para diversos tipos de dispositivos como TVs, *tablets* e *smartphones*. De acordo com o IDC (2022), cerca de 84.1% dos *smartphones* no mundo utilizam o Android, e está previsto que até 2025 esse número aumente, chegando a 84.9% do mercado global.

Uma das principais características que contribuiu para o sucesso do Android foi o seu modelo de código aberto, o qual permite que qualquer pessoa interessada possa baixá-lo e contribuir com o seu desenvolvimento (Android, 2022a). Além disso, os mecanismos de segurança e *kernel* do Android são baseados no Linux, fazendo com que propriedades como memória, processos, segurança de arquivos e redes sejam gerenciados pelo sistema operacional e o torne funcional em diversos dispositivos diferentes. Outra característica importante é que cada aplicativo instalado no Android tem o seu próprio usuário no sistema operacional com as suas devidas restrições de uso, fornecendo assim um ambiente de execução de aplicativos em área restrita (Lecheta, 2013).

2.1.1 Arquitetura do Android

Para que as pessoas desenvolvam suas aplicações com base no sistema operacional Android, uma arquitetura com cinco camadas é oferecida por meio de uma estrutura hierárquica de componentes. A Figura 2.1 apresenta uma visão geral da arquitetura.

Figura 2.1: Visão geral da arquitetura do Android (adaptado de Android (2022b)).

Começando pela camada mais inferior, o *Kernel* do Linux representa uma abstração de *hardware* com um conjunto de *drivers* para serem reutilizáveis, assim como propriedades gerenciáveis de memória, processos e segurança. Além disso, esta camada permite que os desenvolvedores de aplicações Android aproveitem os principais recursos de segurança e que os fabricantes desenvolvam *drivers* de *hardware* para um *kernel* já conhecido, oferecendo maior compatibilidade entre os dispositivos que o utilizam.

Na segunda camada temos a abstração de *hardware*, que consiste em módulos de bibliotecas que implementam regras específicas de acordo com o tipo de *hardware*. Por exemplo, se o acesso à câmera do dispositivo for solicitado, o Android carregará o módulo de biblioteca responsável por este componente de *hardware* (Android, 2022b).

A terceira camada fornece as bibliotecas nativas em C/C++ como Webkit, SSL e SQLite, que proporcionam a maioria das funcionalidades iniciais do Android. Além disso, é nesta camada que o Android *Runtime* (ART) é introduzido. Por meio do ART é possível ter um ambiente execução em área restrita, de forma que todo e qualquer aplicativo tenha acesso apenas ao seu escopo de execução, sem interferir nos demais processos de outras aplicações. O ART foi projetado para executar diversas máquinas virtuais (*Dalvik Virtual Machine - DVM*) em dispositivos de baixa capacidade por

meio da execução de arquivos DEX (*Dalvik Executable* - formato *bytecode*, especialmente projetado para Android), de modo que o mínimo de memória seja consumida. Sendo assim, ao desenvolver aplicativos utilizando linguagem de programação Java[1], um código em formato *bytecode* (`.class`) da linguagem é compilado e posteriormente convertido para o formato `.dex`, para então ser aceito e executado pela máquina virtual (Lecheta, 2013).

A quarta camada, Estruturas Java API, fornece um conjunto completo dos recursos do sistema operacional Android, disponíveis por meio de API ((*Application Programming Interface*), desenvolvidas na linguagem de programação Java. As APIs representam métodos de recursos já desenvolvidos para que o desenvolvedor do aplicativo faça reutilização de componentes e serviços do sistema. Por exemplo, ao criar uma nova tela para o aplicativo, o desenvolvedor pode fazer uso de componentes de tela já existentes no próprio sistema.

A quinta camada, Aplicativos do Sistema, oferece um conjunto de aplicativos que são usados para executar serviços básicos como envio de *e-mail*, SMS (*Short Message Service*), calendário e navegação na Internet. Entretanto, programadores podem desenvolver seus próprios aplicativos para executar estes serviços e substituir os nativos do Android.

Para melhor entendimento do que será abordado, na Seção 2.1.2 apresentamos os tipos de artefatos que caracterizam a estrutura de uma aplicação Android.

2.1.2 Formato e Estrutura de Aplicações Android

Aplicações Android possuem um formato e uma estrutura padrão. O formato corresponde a extensão .APK (*Android Package*), que representa a aplicação final, pronta para ser instalada no dispositivo, enquanto a estrutura dispõe de diversos artefatos de configuração, os quais são apresentados na Figura 2.2.

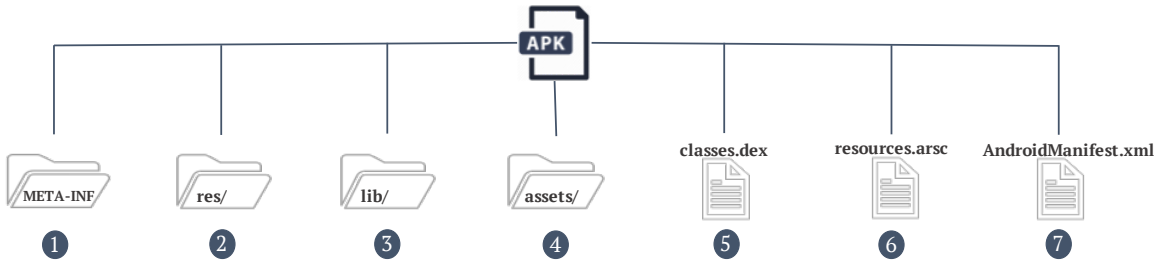


Figura 2.2: Estrutura de uma APK.

[1] https://www.java.com

Na Figura 2.2, o item (1) retrata uma pasta que contém meta informações para a assinatura do aplicativo, como `CERT.RSA`, `CERT.SF` e `Manifest.MF`. O item (2) representa uma pasta com diversas configurações de *layout* gráfico, ícones, imagens e *drawables*. O item (3) consiste em bibliotecas nativas da plataforma. O item (4) representa os recursos não compilados, enquanto que (5) refere-se ao código da aplicação compilado, no formato `.dex`. O item (6) retrata um arquivo que contém recursos de aplicativos pré-compilados e, por fim, o item (7) simboliza o arquivo de configuração geral do aplicativo, o `AndroidManifest.xml` (manifesto), o qual fornece diversas informações como nome das classes, recursos de hardware e software exigidos pelo aplicativo, permissões necessárias para acessar partes protegidas do sistema, filtros de intenção e os componentes utilizados (Android, 2020a).

2.1.3 Componentes de Aplicação

Componentes de aplicação são estruturas pré-definidas para o desenvolvimento de um aplicativo (Android, 2020b) e consiste em quatro tipos diferentes: `Activity`, `Service`, `Content Provider` e `Broadcast Receiver`.

`Activity` representa a interface de usuário da aplicação, ou seja, são as telas que o usuário interage ou visualiza por meio da funcionalidades do aplicativo. `Service` é um componente no Android que permite que uma aplicação execute operações em segundo plano, geralmente usado para executar atividades de operação de longa duração. `Content Provider` é responsável pelo armazenamento e recuperação de dados em aplicativos Android, ou seja, este componente pode ajudar o aplicativo a gerenciar o acesso aos próprios dados armazenados ou aos de outros aplicativos, além facilitar o compartilhamento dessas informações. Por fim, o `Broadcast Receiver`, que é usado para responder a eventos no sistema Android. Os eventos podem ocorrer quando o dispositivo é iniciado, quando uma mensagem ou chamadas são recebidas no dispositivo, ou quando um dispositivo entra no modo avião, por exemplo. Este componente atua diretamente com as intenções (`Intents`) enviadas do sistema operacional ou de outros aplicativos, possibilitando a comunicação entre aplicativos e componentes conforme a solicitação enviada. Por exemplo, caso um aplicativo queira receber uma intenção associada a mensagem SMS, o mesmo terá que declarar o `Broadcast Receiver` no manifesto junto ao filtro de intenção correspondente à ação.

Neste sentido, *malwares* podem fazer uso destas comunicações entre componentes para a realização de ataques de escalonamento de privilégios, onde um aplicativo A com menos privilégio pode invocar componentes de um aplicativo B mais privilegiado para realizar ações maliciosas (Zhongyang et al., 2013). Por conta disto, *malwares* tendem

à declarar mais `Broadcast Receiver` do que os demais componentes (Xu et al., 2016).

2.1.4 Manifesto

O manifesto (`AndroidManifest.xml`) descreve informações essenciais sobre o aplicativo por meio da declaração de seus elementos. Estas informações possibilitam não somente entender as características de funcionamento do aplicativo como também as configurações de segurança da aplicação. Diversos trabalhos utilizaram este artefato como objeto de estudo devido ao rico conjunto de informações que permitem caracterizar aplicações maliciosas (Anwar et al., 2016; Yusof et al., 2017; Hijawi et al., 2017; Feizollah et al., 2017). A Figura 2.3 apresenta uma visão geral do manifesto de um aplicativo.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.keji.danti664">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:label="@string/bookname" android:name="com.keji.danti.MainA"
        android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <receiver android:name="com.android.view.custom.BaseABroadcastReceiver">
            <intent-filter android:priority="2147483647">
                <action android:name="android.net.wifi.PICK_WIFI_WORK"/>
                <action android:name="android.intent.action.UMS_DISCONNECTED"/>
            </intent-filter>
        </receiver>
    </application>
    <uses-permission android:name="android.permission.WRITE_SMS"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_LOGS"/>
</manifest>
```

Figura 2.3: Arquivo de configuração manifesto.

A Figura 2.3 mostra a estrutura do arquivo manifesto. O elemento raiz do arquivo é um atributo para o nome do pacote, representado pelo `package`, cujo o nome é `com.keji.danti664`. Este nome precisa corresponder ao nome do pacote em que o código do aplicativo encontra-se para que funcione corretamente. Além disso, é possível observar alguns componentes de aplicação como `activity` e `broadcast receiver`,

conforme explicados na Seção 2.1.3. A seguir, apresentamos os elementos do manifesto utilizados para o desenvolvimento deste trabalho.

2.1.4.1 Permissões

O modelo de segurança do Android é baseado em permissões (Bergman et al., 2013). O objetivo deste mecanismo é restringir o acesso dos aplicativos aos dados ou recursos confidenciais do *smartphone*. Portanto, se um aplicativo precisa acessar um determinado recurso, a permissão correspondente ao mesmo deve ser definida no manifesto, caso contrário, o acesso não é fornecido. Por exemplo, caso o aplicativo queira ter acesso de leitura à lista telefônica do usuário, o mesmo deve informar a permissão correspondente, que é `READ_CONTACTS`.

Para isto, existem dois tipos de permissões que podem ser declaradas no manifesto, as permissões solicitadas aos usuários (definidas como `uses-permission`) e as permissões personalizadas (definidas como `permission`). O primeiro tipo é mais comum e corresponde às permissões que o aplicativo solicita ao usuário durante o processo de instalação. Entretanto, dependendo da categoria do nível de proteção[2] da permissão, o sistema pode conceder o acesso automaticamente ou solicitar a confirmação do usuário. O segundo tipo de permissão é menos comum e geralmente é criada pelo próprio desenvolvedor do aplicativo de acordo com a necessidade de acesso.

Neste trabalho, iremos estudar apenas o primeiro tipo de permissão devido ser comum em aplicações e por ser considerada objeto de estudo em outros trabalhos da literatura (Wang et al., 2017; Kirubavathi & Anitha, 2018). A Figura 2.4 apresenta a declaração de duas permissões do tipo `uses-permission` no manifesto.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
</manifest>
```

Figura 2.4: Declaração de permissão no manifesto.

A Figura 2.4 apresenta as permissões `CAMERA` e `READ_CONTACTS`, as quais são consideradas perigosas de acordo com o nível de proteção. Por meio de tais permissões, é possível acessar câmera do dispositivo e ler a lista telefônica do usuário, o que torna oportuno aplicações maliciosas obterem proveito de informações confidenciais. Portanto, as permissões podem ser vistas como indicadores de possíveis comportamentos

[2] https://developer.android.com/guide/topics/permissions/overview

maliciosos por meio das aplicações que as solicitam, assim como podem fornecer informações sobre os riscos que apresentam diante concessão dos privilégios fornecidos (Rovelli & Vigfússon, 2014; Wang et al., 2014).

2.1.4.2 Filtro de Intenção

Componentes de aplicativos Android (*activity*, *service*, *broadcast receivers*, *content providers*) são ativados por meio de `Intents`, os quais representam objetos de mensagens trocadas entre si para solicitar uma determinada ação. Para isto, o Android dispõe de dois tipos de *Intents*, o explícito e o implícito.

Os *Intents* explícitos definem o tipo de componente à ser utilizado pela aplicação, enquanto o *Intents* implícitos declaram ações gerais a serem executadas, permitindo que componentes de outras aplicações ou do próprio Android a realize diante a definição fornecida pelo filtro de intenção (definido como `intent-filter`). Por exemplo, caso o usuário queira navegar na Internet por meio de um *browser* e a aplicação esteja utilizando um *Intent* implícito, o Android solicitará do usuário a confirmação de qual *browser* ele deseja utilizar para concluir esta ação. Caso seja um *Intent* explícito, o Android utilizará o componente que foi declarado previamente no aplicativo, ou seja, o *browser* considerado padrão no Android.

A Figura 2.5 apresenta a declaração de um filtro de intenção que define duas ações (definidas como `action`), `SMS_RECEIVED` e `BOOT_COMPLETED`. No âmbito de aplicações maliciosas, estas ações podem ser utilizadas de forma indevida para notificar o *malware*. Por exemplo, `SMS_RECEIVED` permite que o aplicativo que a declarou seja notificado sobre o recebimento de mensagens SMS e caso haja envio de informações sigilosas, o aplicativo poderá ser notificado e, em conjunto com permissões como `READ_SMS`, ler o seu conteúdo. Enquanto isso, `BOOT_COMPLETED` permite que *malwares* sejam notificados sobre o processo de inicialização do Android para iniciar atividades maliciosas (Feizollah et al., 2017).

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</manifest>
```

Figura 2.5: Declaração de *intent-filter* no manifesto.

Sendo assim, o manifesto será utilizado como artefato de estudo por este trabalho, dado o grande conjunto de informações contidas no mesmo e por estar presente em

botnets móveis. Além disso, o manifesto também já foi utilizado em outros trabalhos para caracterizar aplicações maliciosas (Karim et al., 2015; Anwar et al., 2016; Yusof et al., 2017).

2.2 Botnet Móvel

Silva et al. (2013) definem *botnet* como uma rede ou infraestrutura formada por computadores "escravos" (*bots*) e controlados por uma entidade (*botmaster*), cuja a intenção é realizar atividades maliciosas. Os *bots* são aplicações maliciosas executadas em computadores os quais permitem que os *botmasters* os controlem de forma remota a fim de executar diversas atividades fraudulentas. Para realizar tais atividades, o envio de comandos aos *bots* é realizado por meio de servidores de comando e controle (C&C), responsável pela comunicação entre o *botmaster* e os *bots*.

Para que um determinado dispositivo faça parte de uma *botnet*, é necessário passar por um conjunto de etapas que se integram, conforme apresentado na Figura 2.6.

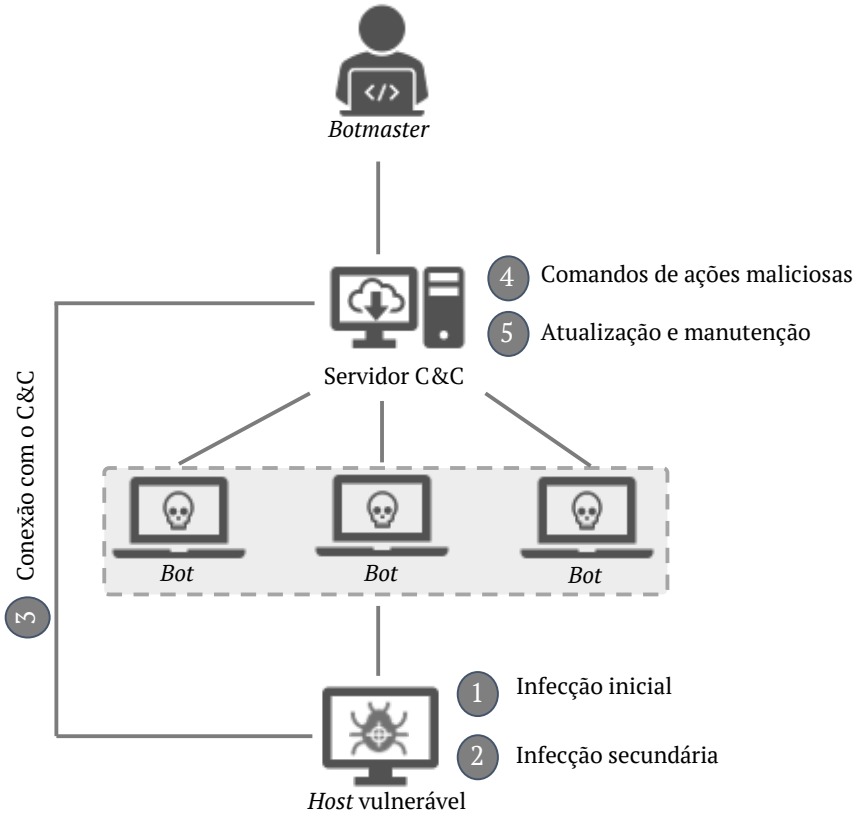


Figura 2.6: Etapas do ciclo de vida de uma *botnet*.

A etapa (1) do ciclo de vida de uma *botnet* consiste na infecção primária de

um dispositivo vulnerável que, a partir de uma aplicação maliciosa, é inicialmente infectado e torna-se um potencial *bot*. Em sequência, a etapa (2) refere-se a infecção secundária, onde outras aplicações maliciosas são baixadas e executadas para fazer do dispositivo um autêntico *bot*. Na etapa (3), uma conexão é estabelecida com o C&C para obter instruções e atualizações. Posteriormente, na etapa (4), o novo *bot* aguarda por comandos que definem as ações maliciosas a serem realizadas. Por fim, a etapa (5) consiste na manutenção e atualização do *malware*, de modo que seja possível manter a sua rede de *bots* ativa e em pleno funcionamento (Silva et al., 2013).

Com o advento de *smartphones* e a sua alta disponibilidade para acesso à Internet, os mesmos tornaram-se um novo meio para propagação de *botnets* (Pieterse & Olivier, 2012). Desta forma, dando origem às *botnets* móveis que, conforme as *botnets* tradicionais, também permitem que o atacante controle o dispositivo da vítima remotamente para realizar atividades maliciosas (Aresu et al., 2015).

Xiang et al. (2011) definem *botnet* móvel comum grupo de *smartphones* comprometidos e controlados remotamente pelo *botmaster* por meio dos servidores de C&C. Para que a comunicação ocorra entre o C&C e os *bots*, os atacantes geralmente utilizam protocolos de comunicação como SMS e HTTP (Anwar et al., 2016). Além disso, diferentemente das *botnets* tradicionais, que possuem comportamento diurno devido aos computadores geralmente serem desligados à noite (Dagon et al., 2006), as *botnets* móveis possuem alta disponibilidade, dado que dificilmente os *smartphones* são desligados, o que torna-se atrativo para a realização de atividades maliciosas. Portanto, considerando tais fatores acerca de *botnets* móveis, torna-se necessário empregar abordagens que auxiliem na mitigação desse tipo de ameaça.

2.3 Abordagens para Análise de Android Botnet

Para realizar o estudo de programas maliciosos que compõem as *botnets*, é importante compreender as abordagens utilizadas para análise. Existem duas abordagens principais para se obter as características e comportamentos dos *malwares*: análise estática e análise dinâmica.

A análise estática consiste em analisar um programa sem realizar a sua execução. É uma abordagem que consome menos recursos e tempo de análise (Wang et al., 2018) comparado às demais, assim como não exige a execução do aplicativo. Além disso, requer menos características para obter uma taxa de falsos positivos satisfatória (Wu et al., 2012), o que é considerado um fator importante no âmbito de análise de aplicações maliciosas. Permissão, filtros de intenção, componentes e chamadas de API são

exemplos de recursos que podem ser observados nesta abordagem (Karim et al., 2015; Anwar et al., 2016; Kirubavathi & Anitha, 2018; Yerima et al., 2021).

Ao contrário da análise estática, a análise dinâmica avalia o programa durante a sua execução, possibilitando identificar padrões característicos de atividades maliciosas durante o tempo de execução do aplicativo. Entretanto, esta abordagem geralmente utiliza um ambiente controlado para a execução das aplicações, necessitando de um maior esforço manual, assim como um consumo computacional maior do dispositivo e consequentemente uma infraestrutura adicional (Wu et al., 2012; Aresu et al., 2015). De acordo com Faruki et al. (2014), restrições de recursos computacionais dos dispositivos limitam a aplicação desta abordagem com êxito. Chamadas de sistema e o tráfego HTTP são exemplos de recursos que podem ser observados nesta abordagem (Aresu et al., 2015; da Costa et al., 2017).

2.4 Técnicas Empregadas

Dada a quantidade crescente de dados, recursos e característicos a serem analisadas em busca de similaridades e padrões em aplicações maliciosas, torna-se necessário o uso de métodos automatizados que, não só aumentem a eficiência do processo como também promovam maior confiabilidade na identificação de padrões em larga escala. Para isto, técnicas de aprendizagem podem ser empregadas. Neste trabalho, consideramos o uso da técnica de aprendizagem de máquina por meio de algoritmos de aprendizado supervisionado.

2.4.1 Aprendizagem de Máquina

Segundo Bishop (2006), Aprendizagem de Máquina (AM) é um processo de reconhecimento de padrões, enquanto Murphy (2012) define como um conjunto de métodos que podem detectar automaticamente padrões nos dados e, assim, ser usado para prever dados futuros ou ajudar na tomada decisões sob incerteza.

Em AM, programas são desenvolvidos para aprender com experiências passadas, conhecidas como hipótese. Para isto, um conceito denominado indução é empregado, tornando-se capaz de obter conclusões genéricas a partir de um conjunto de dados utilizados para realizar o treinamento do algoritmo e posteriormente criar um modelo que será usado para classificação dos dados. Portanto, algoritmos empregados por esta técnica aprendem a induzir a hipótese e tornam-se capazes de resolver um problema a partir dos dados apresentados como amostra de entrada (Faceli et al., 2011). Reconhecimento de atividades de palavras faladas, predição de taxas de cura de pacientes

com diferentes doenças e detecção de fraude de cartão de crédito, são exemplos de aplicações bem sucedidas diante problemas do mundo real (Faceli et al., 2011).

É importante ressaltar que cada algoritmo de aprendizagem de máquina possui uma representação indutiva diferente diante a hipótese. Por exemplo, algoritmos de árvores de decisão usam uma estrutura na qual cada nó interno é representado por uma pergunta diferente, e os nós externos estão associados a uma classe ou atributo alvo. Em contrapartida, redes neurais artificiais representam uma hipótese por um conjunto de valores reais, os quais estão associados aos pesos das conexões de rede (Faceli et al., 2011). Além disso, AM pode ser classificada em quatro modalidades (Marsland, 2014), sendo:

- Aprendizado supervisionado: aprende sobre um determinado conjunto de entrada de dados, ajustando os parâmetros do modelo por meio das respostas obtidas e esperadas. Em outras palavras, o algoritmo busca generalizar corretamente a todas as entradas possíveis com base em exemplos de respostas esperadas.

- Aprendizado não supervisionado: respostas corretas não são fornecidas e o algoritmo tenta identificar padrões por meio das entradas de dados, de modo que os mesmos sejam categorizados de acordo com os padrões similares. Ou seja, os parâmetros do modelo são ajustados com base na maximização da qualidade de resposta obtida.

- Aprendizado por reforço: busca encontrar ações adequadas em uma determinada situação para maximizar a obtenção de recompensa. Na abordagem de aprendizagem por reforço, o algoritmo é informado se a resposta está errada, mas não é informado sobre como corrigi-la. Torna-se necessário explorar e experimentar diferentes possibilidades até descobrir como obter a resposta certa.

- Aprendizado evolutivo: é baseado na evolução biológica, visando a adaptação do mesmo para melhorar as taxas de classificação.

Na literatura existem diversos algoritmos de classificação que podem ser utilizados, como Máquina de Vetores de Suporte (*Support Vector Machine*), K-Vizinhos mais Próximos (*K-Nearest Neighbors - KNN*) e Árvore de Decisão (*Decision Tree*) e Naive Bayes (*Naive Bayes*). Neste trabalho, por meio do aprendizado supervisionado, os algoritmos KNN e Árvore de Decisão foram adotados. A escolha de tais classificadores é baseada na revisão de trabalhos relacionados e revisão sistemática.

2.4.2 KNN

É um algoritmo utilizado para problemas de classificação e uma característica predominante é que o mesmo não constrói um modelo de aprendizado, ou seja, as instâncias de treinamento são guardadas para que a inferência seja feita diante a entrada do novo dado enviado ao algoritmo. Esse estilo de avaliação é conhecida como "avaliação preguiçosa" ("*lazy evaluation*") e o seu processamento pode dispender muito tempo, dependendo do conjunto de instâncias (da Silva et al., 2017).

Portanto, para classificar uma instância desconhecida T, o algoritmo baseia-se no conjunto de K exemplos mais próximos com classes já conhecidas, dentro de um espaço de D características. Tal conjunto a ser observado é dado por métricas distância diante suas similaridades. Geralmente, as métricas aplicadas são a distância *Euclidiana* e *Manhattan* (Goldschmidt et al., 2015). Além disso, é importante ressaltar que, caso o conjunto de K exemplos mais próximos tenham diferentes classes e gere divergência, utiliza-se regra de seleção, como voto da maioria.

A Figura 2.7 apresenta um exemplo com $K = 2$ e $K = 5$, a fim de determinar a classe de uma instância desconhecida T.

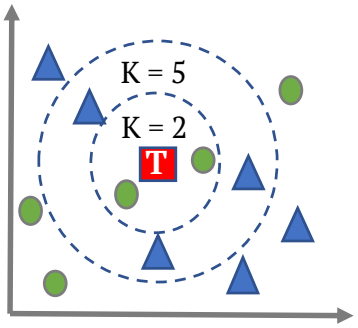


Figura 2.7: Exemplo de decisão utilizando 2-NN e 5-NN.

Neste contexto, da Silva et al. (2017) apresenta o cálculo das métricas de distância. Sendo, a equação da distância de *Manhattan* representada por:

$$dist_{\vec{x_p},\vec{x_q}} = \sum_{j=1}^{d} |x_{pj} - x_{qj}|\,, \tag{2.1}$$

onde, *dist* é a medida de distância, e $\vec{x_p}, \vec{x_q}$, objetos representados por vetores no espaço R^d, à serem observados como espaços geometricamente perfeitos. $\sum_{j=1}^{d}$ é a soma da diferença absoluta de $|x_{pj} - x_{qj}|$, no qual são os elementos do vetor. Em contrapartida, o cálculo da distância *Euclidiana*, representada por:

$$d_{\vec{x_p},\vec{x_q}} = \sqrt{\sum_{j=1}^{d}(x_{pj} - x_{qj})^2}, \tag{2.2}$$

onde, *dist* e $\vec{x_p}, \vec{x_q}$ são equivalentes ao que foi apresentado na equação de *Manhattan*. Entretanto, diferentemente da anterior, esta métrica visa representar a distância física entre os objetos de um espaço d-dimensional, a partir da raiz quadrada da soma dos quadrados, por meio de $\sqrt{\sum_{j=1}^{d}(x_{pj} - x_{qj})^2}$, dada hipotenusa.

2.4.3 Árvore de Decisão

Segundo Faceli et al. (2011), Árvore de Decisão é um algoritmo que utiliza a estratégia de divisão e conquista para resolver um problema de decisão, onde um problema complexo é dividido em problemas mais simples, aos quais recursivamente aplica-se a mesma estratégia. Portanto, as soluções dos subproblemas podem ser combinadas, em forma de árvore, para produzir uma solução do problema maior que os originou. Enquanto isso, Goldschmidt et al. (2015) definem como um modelo de representação de conhecimento em que cada nó interno representa uma decisão sobre um atributo que determina como os dados estão particionados pelos nós filhos. A Figura 2.8 apresenta um exemplo de classificação em forma de Árvore de Decisão.

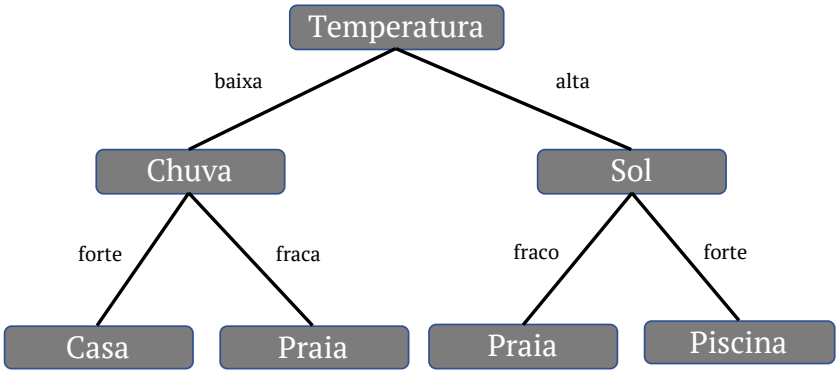


Figura 2.8: Representação de uma Árvore de Decisão.

A Figura 2.8 representa uma Árvore de Decisão que auxilia na tomada de decisão para saber se é um bom dia para ir à praia, ficar em casa ou ir para a piscina, dada a condição do clima e temperatura do dia. Por exemplo, se no dia a temperatura estiver alta, fará sol, e se o sol estiver forte, poderá ir para a piscina.

A construção da árvore ocorre por meio de um algoritmo que, de forma iterativa, analisa todos os atributos correspondentes ao conjunto de dados previamente rotulados, permitindo assim o processo de aprendizado do modelo de classificação. Entretanto,

é importante ressaltar que, quanto maior a árvore, maior será o número de regras que derivam da mesma. Sendo assim, uma árvore com muitos nós resulta em modelo que pode ficar prejudicado na sua generalização, dada a quantidade de comparações à serem realizadas (da Silva et al., 2017).

De forma geral, a Árvore de Decisão é um modelo que corresponde a um conjunto de regras que auxiliam na tomada de decisão e descoberta de conhecimento, dado que a representação de tais aspectos podem ser facilmente entendidas.

2.4.4 Validação Cruzada

Para avaliar a eficácia dos modelos de classificação diante a sua capacidade de generalização, é necessário usar uma técnica de validação de modelo. A utilização de tais técnicas auxiliam a evitar situações de *overfitting* e *underfitting*. A primeira ocorre quando o modelo se ajusta bem aos dados de treinamento e a segunda ocorre quando o modelo não é capaz de capturar nenhum padrão de tendência sobre os dados de treinamento. Ambas podem levar a uma baixa eficácia quando aplicadas a novos dados. Nesta seção, apresentamos a técnica de validação de modelo utilizada por este trabalho, a validação cruzada.

A validação cruzada, também conhecido como *k-fold*, é um técnica de reamostragem usada para validar precisão do modelo (Geisser, 1974). A técnica consiste em dividir aleatoriamente um conjunto S em k subconjuntos exclusivos S_1, S_2, ..., S_k de tamanho aproximadamente igual (Kohavi et al., 1995). Neste sentido, o modelo é treinado usando $k - 1$ subconjuntos com dados de treinamento e o modelo resultante é validado no subconjunto restante dos dados. O resultado é dado pela média dos valores calculados após todas as interações do modelo diante os subconjuntos. Dependendo da quantidade de dados e subconjuntos criados, essa abordagem pode ser computacionalmente cara, mas pode fornecer bons resultados ao avaliar a estabilidade da predição do modelo, dado que para cada subconjunto serão utilizados diferentes dados para treino e teste.

2.5 Considerações Finais

Este capítulo apresentou um resumo dos conceitos teóricos necessários para o entendimento deste trabalho, começando com o sistema operacional Android, sua arquitetura, estrutura de aplicativos Android e elementos de configuração presentes no arquivo de manifesto que serão utilizados como objetos de estudo. Em seguida, foram explicados os conceitos sobre *botnets* móveis, assim como as abordagens empregadas para realizar

análise de deste tipo de ameaça visando as vantagens e desvantagens de cada abordagem. Posteriormente, apresentamos as técnicas e algoritmos utilizados no âmbito de aprendizagem de máquina, assim como descrevemos os que serão utilizados por este trabalho e como avaliar a eficácia dos modelos de classificação.

Capítulo 3

Trabalhos Relacionados

Neste capítulo será apresentada uma revisão da literatura. Para isto, é realizada uma classificação dos trabalhos em duas seções. A Seção 3.1 aborda os trabalhos sobre detecção de *malware* e a Seção 3.2 apresenta os trabalhos de detecção de *botnet* móvel. Por fim, na Seção 3.3 é apresentado um comparativo entre os trabalhos, considerando os tipos de recursos adotados, algoritmos de aprendizagem de máquina e a abordagem empregada, assim como as considerações finais do capítulo.

3.1 Detecção de Malware

Diversos estudos sobre detecção de *malwares* foram conduzidos nos últimos anos e várias abordagens baseadas em permissões do Android foram propostas. Por exemplo, Sanz et al. (2013) apresentam o PUMA, um método totalmente baseado em permissões do Android. Os autores consideram diversos algoritmos de aprendizado de máquina para realizar a classificação binária, ou seja, determinar se a aplicação é benigna ou maliciosa. O melhor resultado foi obtido do algoritmo Floresta Aleatória. Similarmente, Samra et al. (2013) também utilizam permissões, mas a classificação ocorre por meio do algoritmo do clusterização *K-Means*.

Neste sentido, Şahın et al. (2018) propõem uma abordagem baseado em permissão utilizando o método de ponderação *Relevance Frequency* (RF), o que permitiu definir a importância de cada característica com base na análise estatística. Para conduzir a classificação das amostras, os autores utilizaram os algoritmos KNN e *Naive Bayes*. O melhor resultado foi obtido pelo KNN, com 93,27% de precisão. A limitação acerca de tais abordagens empregada por Sanz et al. (2013), Samra et al. (2013) e Şahın et al. (2018) é que apenas utilizar permissão pode não ser o suficiente para detectar *malwares*,

assim como pode gerar uma taxa alta de falso positivo, sendo assim necessário utilizar outros meios para caracterizar tais ameaças (Aafer et al., 2013; Huang et al., 2013).

Para atender este problema, Wu et al. (2012) propõem o método DroidMat que, além de permissões, também consideram filtros de intenção, componentes de aplicação e chamadas de API. Para realizar a classificação, os autores utilizam diferentes combinações entre os conjuntos de características e algoritmos de aprendizado de máquina como KNN, *K-Means* e *Naive Bayes*.

Similarmente, Arp et al. (2014) apresentam o método Drebin. Além de filtros de intenção, permissões, componentes de aplicação, componentes de *hardware* e chamadas de API, os autores adicionaram IPs, o qual é comum em aplicações que se comunicam com C&C. Entretanto, IPs do C&C podem mudar com novas atualizações (Gezer et al., 2019) e, assim, prejudicar a assertividade das amostras. Para conduzir a classificação, os autores utilizaram o algoritmo SVM.

Ao contrário de Wu et al. (2012) e Arp et al. (2014), nosso método considera apenas permissões e filtros de intenção para representar as características das amostras, com o objetivo de proporcionar maior assertividade na detecção.

3.2 Detecção de Botnet Móvel

Aplicações maliciosas projetadas para transformar *smartphones* em *bots* e compor *botnets* representam uma séria ameaça e oferecem maiores riscos aos usuários e empresas. Neste sentido, alguns estudos possuem como o foco a detecção de Android *botnet* por meio do uso de algoritmos de aprendizagem de máquina. Por exemplo, Yusof et al. (2017) apresentam um estudo para a classificação de Android *botnet* baseado características extraídas a partir de permissões e chamadas de API. Os algoritmos de classificação utilizados foram *Naive Bayes*, KNN, Árvore de Decisão, Floresta Aleatória e *Support Vector Machine* (SVM).

Hijawi et al. (2017) propõem um *framework* para detectar Android *botnet* usando características a partir das permissões e níveis de proteção. Os autores utilizaram algoritmos aprendizado de máquina populares como Floresta Aleatória, Árvore de Decisão, *Multi Layer Perceptron* e *Naive Bayes*. O nível de proteção das permissões podem mudar com novas atualizações do Android, o que o torna um recurso sensível a mudanças e pode gerar viés positivo na fase de treinamento e teste para novas amostras.

Similarmente, Hojjatinia et al. (2020) apresentam um método para detectar Android *botnet* baseado em Redes Neurais Convolucionais (CNN). Para realizar a classificação, os aplicativos foram representados como imagens com base na coocorrência

de permissões utilizadas pelas amostras. O método proposto pelos autores obteve uma acurácia de 97,2%, que é um resultado promissor considerando que apenas permissões foram utilizadas no estudo. Entretanto, conforme será visto no Capítulo 5, com a adição de um novo conjunto de características é possível identificar comportamentos discriminativos sobre as aplicações assim como obter resultados superiores.

Kirubavathi & Anitha (2018) apresentam um *framework* de detecção de Android *botnet* com base nas permissões e recursos de *hardware* e *software* usados pela aplicação. As informações são extraídas a partir do arquivo *AndroidManifest.xml*. Os autores argumentam que a detecção de *botnets* pode ser realizada por meio da correlação das permissões solicitadas e dos recursos de *hardware* e *software* usados. Por exemplo, um aplicativo de *botnet* que envia mensagens SMS pode conter a permissão `SEND_SMS` e o componente de *hardware* `android.hardware.telephony`. O *framework* utiliza o algoritmo de mineração de regras de associação Apriori para extrair as combinações significativas das permissões solicitadas e recursos usados. Para classificar, os autores utilizaram *Naive Bayes*, SVM e *REPTree*. Apesar da alta taxa de detecção, recursos de *hardware* e *software* nem sempre estão disponíveis ou são utilizados entre as aplicações, o que pode ser um fator limitante ao tentar extrair essas informações e, consequentemente, gerar dependência de tais características para classificar as amostras.

Em um estudo mais recente, Yerima & Alzaylaee (2020) propôs uma abordagem para detecção de Android *botnet* com base na extração de chamadas de API, permissões, comandos de execução, extensão de arquivos e filtros de intenção. Para classificar as amostras, os autores utilizaram algoritmos tradicionais, assim como Redes Neurais Convolucionais, onde obtiveram o melhor resultado. Entretanto, o espaço de características é demasiado, o que aumenta desnecessariamente a complexidade do modelo de classificação.

3.3 Considerações Finais

A Tabela 3.1 apresenta uma visão geral do comparativo realizado do nosso trabalho com os trabalhos da literatura, incluindo os tipos de características, algoritmos e a abordagem empregada. Diferentemente dos demais trabalhos apresentados, propomos um método que considera classificação por meio de dois conjuntos de características que oferecem alta taxa de assertividade e baixo valor de falso positivo, conforme será visto no Capítulo 5.

Tabela 3.1: Trabalhos relacionados.

Trabalho	Características	Algoritmos	Abordagem
Sanz et al. (2013)	Permissões	*Simple Logistic*, *Naive Bayes*, *Bayes Network*, *Sequential Minimal Optimization*, IBK, J48, *Random Forest*	Detecção de *malware*
Samra et al. (2013)	Permissões	*K-Means*	Detecção de *malware*
Wu et al. (2012)	Permissões, filtros de intenção, componentes de aplicação, chamadas de API	KNN, *K-Means*, *Naive Bayes*	Detecção de *malware*
Şahın et al. (2018)	**Permissões**	**SVM**	**Detecção de *malware***
Arp et al. (2014)	Permissões, filtros de intenção, componentes de aplicação, chamadas de API restritas, chamadas de API suspeitas, recursos de *hardware* e *software*, endereços de rede,	SVM	Detecção de *malware*
Yusof et al. (2017)	Permissões, chamadas de API	*Naive Bayes*, KNN, *Decision Tree*, *Random Forest*, SVM	Detecção de *botnet* móvel
Hijawi et al. (2017)	Permissões, níveis de proteção	*Random Forest*, *Decision Tree*, *Multi Layer Perceptron*, *Naive Bayes*	Detecção de *botnet* móvel
Kirubavathi & Anitha (2018)	**Permissões e recursos de *hardware* e *software* usados pela aplicação**	***Naive Bayes*, SVM e *REPTree***	**Detecção de *botnet* móvel**
Yerima & Alzaylaee (2020)	Permissões, chamadas de API, comandos de execução, extensão de arquivos e filtros de intenção	CNN	Detecção de *botnet* móvel
Hojjatinia et al. (2020)	**Permissões**	**CNN**	**Detecção de *botnet* móvel**
Nosso trabalho	Permissões e filtros de intenção	KNN, *Decision Tree*	Detecção de *botnet* móvel

Neste capítulo apresentamos uma revisão dos trabalhos relacionados a detecção de *malwares* e *botnets* móveis. Ao analisarmos os trabalhos, foi visto que existe oportunidade de investigar a utilização de apenas permissões e filtros de intenção como conjunto de características, observando o impacto em eficácia em relação aos métodos existentes. Além disso, este trabalho atende o problema de classificação por meio de

um método de classificação, empregando a combinação de diferentes características com base em critério de relevância e algoritmos de aprendizado de máquina tradicionais. No próximo capítulo, será apresentada a solução proposta por este trabalho.

Capítulo 4

Método Proposto

Este capítulo descreve o método proposto para detecção de Android *botnet*. A Figura 4.1 apresenta uma visão geral do método, composto por 4 etapas: descompilação, extração de características, seleção de características e detecção das amostras de Android *botnet*, as quais serão explicadas com mais detalhes a seguir.

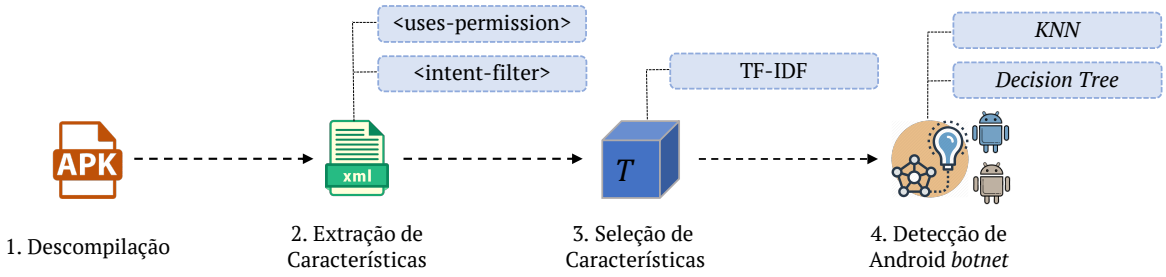


Figura 4.1: Método para detecção de Android *botnet*.

4.1 Descompilação

A primeira etapa, chamada de descompilação, recebe como entrada um aplicativo Android. Cada aplicativo corresponde a extensão .APK, que representa a aplicação final e inclui todo o código fonte, assim como um conjunto de artefatos, dentre os quais encontra-se no arquivo manifesto (manifest.xml). Este artefato fornece diversas informações sobre o aplicativo como permissões solicitadas pela aplicação, versão do aplicativo, filtros de intenção e outros componentes conforme descritos na Seção 2.1.4. Para obter tais informações, o manifesto de cada aplicativo foi extraído por meio da ferramenta APKTool[1] e posteriormente utilizado na etapa subsequente, para realizar

[1] https://ibotpeaches.github.io/Apktool/

a extração de características.

4.2 Extração de Características

De posse do arquivo de manifesto, a segunda etapa corresponde a extração das características presentes neste artefato, obtido de cada aplicativo. Especificamente, nós observamos as permissões solicitadas ao usuário, definida por meio de `uses-permission`, e as ações à serem realizadas pelos componentes da aplicação, definidas por meio do `intent-filter`. As permissões fornecem privilégios de acesso à dados e recursos da aplicação como acesso à Internet, localização do usuário e lista de contatos, enquanto os filtros de intenção especificam as ações realizadas por meio dos componentes utilizados no Android, conforme explicado na Seção 2.1.4.

Um dos fatores que contribuíram na definição de escolha de `uses-permission` e `intent-filter` foi a representatividade de cada um desses recursos presentes no arquivo de manifesto. Para obter a representatividade, um `script` foi desenvolvido para identificar a frequência de cada recurso presente no manifesto. Por exemplo, a Tabela 4.1 destaca a representação individual dos recursos disponíveis na base de dados (5.1.1), sendo `uses-permission` e `intent-filter` os recursos que apresentam maior representatividade.

Tabela 4.1: Representatividade dos recursos em todo o conjunto de amostras.

Índice	**Recurso**	**Representatividade**
1	`uses-permission`	50.54%
2	`intent-filter`	27.99%
3	`service`	7.76%
4	`receiver`	7.65%
5	`meta-data`	4.37%
6	`uses-feature`	0.87%
7	`permission`	0.53%
8	`provider`	0.29%

É importante destacar que ações do filtro de intenção como `MAIN` e `VIEW`, que são

comuns a todas as aplicações por representarem ações genéricas[2] foram desconsideradas para não prejudicar a análise e o estudo realizado. A partir disso, com o conjunto de características obtido, a próxima etapa realiza a seleção das características mais relevantes para o conjunto de aplicações benignas e de Android *botnet*.

4.3 Seleção de Características

Para realizar a seleção de características, utilizamos o *Frequency–Inverse Document Frequency* (TF-IDF), que é uma medida de avaliação presente no campo de processamento de recuperação da informação (Hiemstra, 2000), responsável por determinar a relevância de uma palavra (termo) para um texto (documento), onde ao final um valor (peso) é atribuído de acordo com a importância do termo diante o seu número de aparições nos documentos.

O TF corresponde ao número de vezes que um termo ocorre em um documento, dividindo o seu peso pelo total de termos do documento, onde o peso varia de acordo com o tamanho do documento analisado. Em contrapartida, o IDF busca medir a relevância do termo em relação ao conjunto total de documentos por meio do logaritmo do número total de documentos, dividido pelo número de documentos em que o termo aparece (Ramos et al., 2003; Tata & Patel, 2007). Por exemplo, dado um documento d com 1.000 termos, onde o termo t "`CAMERA`" ocorre 50 vezes, tem-se a seguinte representação:

$$TF(CAMERA) = \frac{50}{1.000} = 0.05 \tag{4.1}$$

Em sequência, considerando que existam 500 documentos no conjunto de amostras e que o termo "`CAMERA`" aparece em 30 destes, tem-se então:

$$IDF(CAMERA) = \log \frac{500}{30} = 1.22 \tag{4.2}$$

Desta forma, o TF-IDF do termo "`CAMERA`" é representado por:

$$TF\text{-}IDF(CAMERA) = TF(CAMERA) * IDF(CAMERA) = 0.05 * 1.22 = 0.06 \tag{4.3}$$

O resultado final do TF-IDF é um peso individual para cada termo calculado. Este peso varia entre 0 e 1, e significa dizer que, quanto mais próximo de 0 menor é a relevância do termo, e quanto mais próximo de 1, maior é a relevância.

[2] https://developer.android.com/reference/android/content/Intent

No contexto deste trabalho, as amostras de Android *botnet* são tratadas como um conjunto de documentos D, onde cada amostra é representada como um documento d_i. Portanto, considerando n amostras de Android *botnet*, temos o conjunto $D = \{d_1, d_2, ..., d_n\}$. Além disso, para cada documento, temos um conjunto de T termos, simbolizando as características extraídas de cada amostra, onde cada termo t_j está contido no vetor $T = \{t_1, t_2, ..., t_m\}$. A mesma representação vale para as amostras benignas.

A Tabela 4.2 apresenta um exemplo com 03 amostras distintas diante os termos que possuem. Por exemplo, o termo t_1 do documento d_1 possui o peso atribuído pelo TF-IDF de 0,510089, o qual indica ser mais relevante para d_1 do que para d_2 ou d_3.

Tabela 4.2: Exemplo de TF-IDF para cada termo do documento.

Documento	**Termo**		
	t_1	t_2	t_3
d_1	0,510089	0,389095	0,069652
d_2	0,353369	0,336011	0,240945
d_3	0,368152	0,346282	0,181251

Com os pesos do TF-IDF calculados, as características com os maiores valores são consideradas para compor o vetor de características de cada amostra de Android *botnet*. O Algoritmo 1 foi desenvolvido para auxiliar nesta atividade.

Algoritmo 1 Seleção de termos mais relevantes

```
1: Entrada: Todas as amostras D, Quantidade de termos relevantes x
2: Saída: Conjunto dos x termos mais relevantes
3: T = Extração de todos os termos t das amostras em D
4: TF_IDF = calcula_TF_IDF(D, T)
5: para cada amostra d ∈ D faça
6:     t_maior_pontuacao = t_maior_pontuacao_do_TF_IDF(TF_IDF, d.termos, x)
7:     para t ∈ t_com_maior_pontuacao faça
8:         se t ∉ termos_relevantes então
9:             termos_relevantes.adiciona(t, classe)
10:        senão
11:            continua
12:        fim se
13:    fim para
14: fim para
15: termos_relevantes = filtrar_relevantes_por_classe(termos_relevantes)
16: retorna termos_relevantes
```

O Algoritmo 1 busca para cada amostra d_i uma quantidade x de características (termos) mais relevantes baseado na lista de todos os termos T, representados pelo

peso do TF-IDF. Caso o termo t_j não esteja presente na lista de termos relevantes, o mesmo é adicionado. Por fim, é possível agrupar apenas os termos mais relevantes de acordo com quantidade desejada para cada classe. Neste trabalho, os 15 termos mais relevantes para cada classe foram selecionados, onde valor quantitativo foi definido por meio de um estudo empírico em que avaliações com 5, 10, 15, 20, 25 e 50 termos foram realizadas. Entretanto, os melhores resultados foram obtidos por meio da seleção dos 15 termos mais relevantes presentes na base de dados utilizada por este trabalho (5.1.1).

De posse do vetor de características com os termos mais relevantes, na etapa seguinte é realizada a detecção das amostras de Android *botnet*.

4.4 Detecção de Android Botnet

Nesta etapa cada amostra é classificada com base no vetor de caraterísticas. Portanto, cada característica no vetor é representada pelo peso do TF-IDF.

Por meio deste vetor de características, algoritmos de aprendizado de máquina supervisionado tradicionais podem ser utilizados para gerar o modelo de detecção, assim como algoritmos de aprendizado profundo como redes neurais de convolução e redes neurais *autoencoders* também podem ser empregados. Entretanto, para o escopo deste trabalho, apenas os algoritmos KNN e Árvore de Decisão foram considerados devido sua ampla adoção na literatura (Anwar et al., 2016; Yusof et al., 2017; Kirubavathi & Anitha, 2018), conforme apresentado no Capítulo 3.

4.5 Considerações Finais

Neste capítulo foi apresentado o método proposto para a detecção de Android *botnet*. A Figura 4.1 apresentou uma visão geral do método, que compreende 4 etapas: descompilação, extração de características, seleção de características e detecção de Android *botnet*. A segunda e terceira etapas são os diferenciais do nosso método em relação aos trabalhos da literatura. A segunda etapa corresponde a utilização de recursos presentes no manifesto como permissões e filtros de intenção, que auxiliam no melhor entendimento de comportamentos maliciosos, enquanto a terceira etapa contribui para a redução da dimensionalidade do espaço de recursos com base em características mais relevantes por meio do TF-IDF, o que aumenta a eficiência dos algoritmos de aprendizado de máquina.

No capítulo 5 apresentaremos os experimentos e resultados diante o problema que buscamos resolver, considerando também a utilização do método proposto para

classificação de famílias de Android *botnet*. Além disso, apresentamos as avaliações relacionadas aos resultados obtidos por meio do método proposto.

Capítulo 5

Experimentos e Resultados

Este capítulo descreve os experimentos e resultados alcançados. O capítulo inicialmente detalha o protocolo experimental empregado, incluindo os algoritmos de aprendizado de máquina, a descrição das bases de dados, o pré-processamento dos dados e os trabalhos da literatura utilizados como *baselines* para comparação com o nosso método proposto, assim como as medidas de avaliação dos classificadores. Por fim, nós apresentamos os experimentos e as considerações finais.

5.1 Protocolo Experimental

Para avaliar a capacidade do método proposto, nós adotamos algoritmos de aprendizado de máquina como KNN e Árvore de Decisão. Além disso, consideramos dois experimentos e uma base de dados de amostras benignas e maliciosas, composta por 1.148 aplicativos benignos e 1849 aplicativos maliciosos.

No primeiro experimento, considerando as amostras benignas e maliciosas, nós avaliamos o desempenho do nosso método proposto em detectar aplicações de Android *botnet* usando a combinação das características mais relevantes extraídas das permissões solicitadas (`uses-permission`) e dos filtros de intenção (`intent-filter`). Para efeito de comparação, nós adotamos os trabalhos proposto por Şahın et al. (2018); Kirubavathi & Anitha (2018); Hojjatinia et al. (2020). No segundo experimento, utilizamos apenas as amostras maliciosas e realizamos um estudo exploratório das principais características de famílias de Android *botnet*, assim como a classificação por meio dos algoritmos de aprendizado de máquina anteriormente mencionados, com o intuito de avaliar a generalização e aplicabilidade do nosso método proposto diante diferentes cenários.

Em ambos os experimentos, 70% das amostras foram utilizadas para treino de 30% foram utilizadas para teste. Para efetuar essa divisão percentual, foi utilizada a biblioteca Scikit-learn[1] da linguagem Python, que possui a função `train-test-split()`, capaz de escolher de forma aleatória as amostras para o treino e teste. Além disso, aplicamos o processo de *Grid Search*, também disponível na biblioteca Scikit-learn, com o intuito de ajustar os hiperparâmetros modelos para otimizar o resultado. Os hiperparâmetros utilizados são apresentados na Tabela 5.1 e estão divididos por tipo de experimento, sendo I para detecção de Android *botnet* e II para classificação de família de Android *botnet*.

Tabela 5.1: Hiperparâmetros dos algoritmos utilizados.

Experimento	**Algoritmo**	**Melhores hiperparâmetros**
I	Árvore de Decisão	*criterion* = *entropy* *max_ depth* = 12 *splitter* = *random*
I	KNN	*algorithm* = *auto* *metric* = *manhattan* *n_ neighbors* = 5 *weights* = *distance*
II	Árvore de Decisão	*criterion* = *entropy* *max_ depth* = 5 *splitter* = *best*
II	KNN	*algorithm* = *auto* *metric* = *manhattan* *n_ neighbors* = 7 *weights* = *distance*

[1] https://scikit-learn.org/

Na Tabela 5.1 podemos verificar que, dentre os hiperparâmetros selecionados, a Árvore de Decisão mede a qualidade da divisão entre os nós da árvore utilizando a entropia e regulariza a árvore limitando sua profundidade à 12 para o primeiro experimento, enquanto para o segundo experimento o limite de profundidade é 5, o que previne o ajuste excessivo. O KNN, por sua vez, utiliza a métrica de distância *manhattan* para ambos os experimentos, assim como define 5 para o primeiro experimento e 7 para o segundo experimento como os vizinhos mais próximos da amostra a ser classificada, de forma que o rótulo desta amostra seja definido ponderadamente de acordo com a proximidade dos vizinhos selecionados.

5.1.1 Base de Dados

As avaliações conduzidas neste trabalho são baseadas em um conjunto de amostras composta por aplicativos benignos e maliciosos coletados do mundo real. O conjunto de amostras benignas é conhecido como *CICInvesAndMal2019* (Taheri et al., 2019) e consiste em 1.150 aplicativos. Em contrapartida, o conjunto de amostras maliciosas é conhecido como *ISCX Android Botnet* (Abdul Kadir et al., 2015) e consiste em 1.929 aplicativos maliciosos agrupados em 14 famílias de *malwares* diferentes. A Tabela 5.2 apresenta uma visão geral da base de dados de aplicativos maliciosos, incluindo o nome da família, quantidade de amostras, o tipo de C&C e algumas características relacionadas às técnicas de infecção como *drive-by download, repackaged application e trojan*.

Tabela 5.2: Visão geral da base de dados de famílias de Android *botnet*.

Família	**Amostras**	**C&C**	**Características**
Bmaster	6	HTTP	roubo de dados, *repackage applications*
RootSmart	28	HTTP	roubo de dados, *repackaged applications*
Sandroid	44	SMS	*ransomware*, *trojan*
NotCompatible	76	HTTP	*drive-by download*, *exploit*
Zitmo	80	SMS	*malware* bancário, engenharia social
Pletor	84	SMS/HTTP	*ransomware*, *trojan*
TigerBot	96	SMS	roubo de dados, *trojan*
Wroba	100	SMS/HTTP	*malware* bancário, *trojan*

Continua na próxima página

Tabela 5.2 – *Continuação da página anterior*

Família	**Amostras**	**C&C**	**Características**
MisoSMS	100	Email	roubo de dados, *trojan*
NickySpy	199	SMS	roubo de dados, *repackaged applications*
AnserverBot	243	HTTP	roubo de dados, engenharia social
PJapps	244	HTTP	*repackaged applications*, *trojan*
Geinimi	264	HTTP	roubo de dados, *repackaged applications*
DroidDream	363	HTTP	roubo de dados, *repackaged applications*

Para os experimentos nós descartamos 80 amostras maliciosas (aplicativos) e 02 amostras benignas. O motivo para descarte é que 02 amostras maliciosas e 02 amostras benignas apresentaram erro durante o processo de descompilação, enquanto as demais 78 amostras maliciosas foram descartadas devido a baixa representatividade, dado que apenas famílias de *malware* com 50 amostras ou mais foram consideradas para não prejudicar o estudo realizado. A Tabela 5.3 apresenta a quantidade de amostras de aplicativos benignos e maliciosos após o processo de descarte.

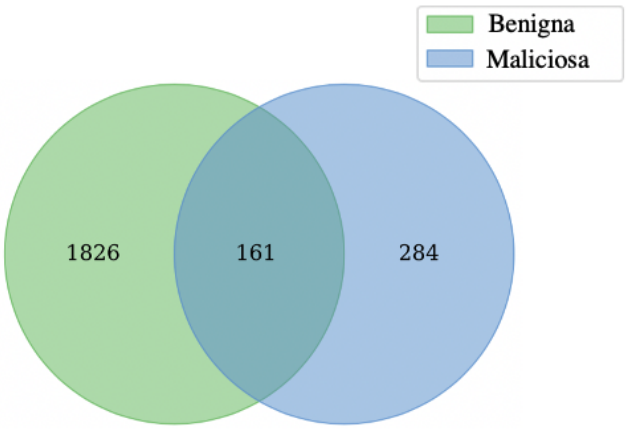
Tabela 5.3: Base de dados de amostras benignas e maliciosas.

Amostra	**Quantidade**
Benigna	1.148
Maliciosa	1.849

5.1.2 Pré-processamento

Neste trabalho, foram considerados os conjuntos de características extraídos a partir das permissões solicitadas (`uses-permission`) e das ações definidas entre os componentes da aplicação por meio de filtros de intenção (`intent-filter`). Entretanto, as ações do filtro de intenção como `MAIN` e `VIEW`, que são comuns a todas as aplicações por representarem ações genéricas, foram descartadas para não prejudicar a análise e o estudo realizado. Como resultado desta etapa, 40.373 permissões solicitadas e 19.964 ações dos filtros de intenção foram obtidos a partir do pré-processamento da base de dados adotada nos experimentos.

Posteriormente, os dados foram analisados, o que nos permitiu ter um melhor entendimento da representatividade de tais conjuntos de características. Neste caso, apesar do número de permissões extraídas ser maior que o número de ações dos filtros de intenção, após realizar uma análise destes conjuntos, foram identificadas 309 permissões únicas, enquanto o número de ações dos filtros de intenção foi superior, chegando ao total de 2.271 ações únicas, conforme apresentado na Figura 5.1. Isso mostra que ao considerar apenas as permissões únicas para compor o conjunto de características, o número de ações torna-se superior, o que pode auxiliar no processo de caracterização das amostras.



(a) Benigna e Maliciosa

Figura 5.1: Resultado do comparativo de `intent-filter` entre aplicativos benignos e maliciosos.

A Figura 5.1 mostra que maior quantidade de ações estão associadas aos aplicativos benignos, o que pode contribuir na distinção dos aplicativos. Para exemplificar características das amostras presentes no conjunto dos filtros de intenção, é possível destacar as ações em comum entre os dois grupos de amostras, como `WIFI_STATE_CHANGED`, que indica que o status do *wi-fi* está ativado, desativado ou é desconhecido e, `PHONE_STATE`, que corresponde a mudança de estado da chamada no dispositivo. Entre as ações exclusivas, foi possível identificar `SENT_SMS, CANCEL_SMS_NOTIFICATION, DOWNLOAD_COMPLETED` presente nas amostras maliciosas, enquanto nas amostras benignas, ações como `ALARM_REMINDER_SPORTS, CHECK_ALARM, ABOUT_APP` foram identificadas.

Esse mesmo tipo de análise foi aplicada para o conjunto de permissões de cada grupo de amostras e permitiu ter um melhor entendimento das características de cada grupo de amostra. Além disso, auxiliou na tratativa dos dados para não prejudicar a análise do estudo realizado. Com o conjunto de características obtido, foi realizada a se-

leção das características mais relevantes para cada conjunto e posteriormente utilizada nos experimentos para validar o nosso método proposto.

5.1.3 *Baselines*

Para avaliarmos o nosso método proposto, adotamos três trabalhos como *baselines* para efeitos de comparação. Entretanto, é importante destacar que as bases de dados e os códigos desenvolvidos pelos autores dos baselines não foram fornecidos, sendo assim necessário realizarmos a implementação dos mesmos conforme descritos em seus trabalhos. No primeiro trabalho, Şahın et al. (2018) apresentam um método de ponderação baseado na frequência de relevância (RF) das permissões presentes nas amostras maliciosas e benignas. Neste caso, para determinar a relevância de uma característica, seu peso é calculado pela equação $\log_2(1 + \frac{a}{c})$, onde a é o número de aplicativos maliciosos que contém a permissão, e c é o número de aplicativos benignos que a contém também. Caso a permissão esteja apenas em uma classe (maliciosa ou benigna), indica que a permissão pode ser discriminante o suficiente para classificação. Se a permissão aparecer em todas as categorias com a mesma frequência, indica que é de baixa relevância. A frequência de relevância para as permissões gera uma matriz esparsa que é utilizada como entrada para aos classificadores. Para realizar a classificação das amostras, os autores utilizaram uma base de dados balanceada com 399 aplicações e dividiram as amostras aleatoriamente entre 70% para treino e 30% para teste, assim como aplicaram validação cruzada de 10 partições. O melhor resultado foi 92,43% de acurácia por meio do algoritmo KNN.

No segundo trabalho, Kirubavathi & Anitha (2018) apresentam um *framework* que utiliza algoritmo de mineração de regras de associação Apriori para extrair as combinações significativas das permissões solicitadas e recursos de *hardware* solicitados para uso. Para isto, os autores realizam análise de frequência das permissões e recursos de *hardware* utilizados pelas aplicações benignas e maliciosas, e posteriormente fazem análise de padrão das características para obter combinações pares de permissão e recursos de *hardware* com base número de aplicações que contém Rp,Uf no conjunto *Dbot* pelo número total de aplicações em *Dbot*, onde Rp são as permissões solicitadas `(uses-permission)` e Uf `(uses-feature)` são os recursos de *hardware* utilizados, e *Dbot* é o conjunto que contém permissões solicitadas e recursos de *hardware* em n aplicações de Android *botnet*. Com os pares de características obtidos, é feito o levantamento de padrões únicos, onde é aplicada a seleção de características com base no algoritmo *Information Gain* (IG). Por fim, 25 conjuntos são definidos de acordo um *ranking* estabelecido pelos valores do IG. Dado os conjuntos estabelecidos, a classifi-

cação ocorre por meio dos algoritmos SVM, *Naive Bayes* e *REPTree*, onde o melhor resultado foi obtido pelo SVM, com 99,10% de acurácia e 2,61% de falso positivo.

Por fim, no terceiro trabalho, Hojjatinia et al. (2020) apresentam um método para detectar Android *botnets* por meio de Redes Neurais Convolucionais (CNN). A CNN proposta é um classificador binário que utiliza as aplicações benignas e maliciosas com base nas permissões utilizadas pelas amostras. O resultado final desta representação é uma imagem em preto e branco, que indica as permissões presentes em cada aplicação, representada por uma matriz $n \times n$, onde os elementos $[i, j]$ apresentam a co-ocorrência das permissões $i\widehat{t}$ e $j\widehat{t}$ de cada aplicativo. Se ambas permissões forem usadas pelo aplicativo, o elemento $[i, j]$ é representado pelo valor 0, caso contrário, o valor 255 é atribuído, e assim é feito para cada uma das permissões contidas em cada aplicativo, sendo a sua representação final uma imagem. Para validar o método proposto, os autores utilizaram 5.450 amostras, sendo 1.800 aplicações maliciosas e 3.650 aplicações benignas. Os resultados mostram uma acurácia de 97,20%, o que é um resultado promissor considerando que apenas permissões foram usadas no estudo.

5.1.4 Medidas de Avaliação

Para avaliar quantitativamente os resultados dos modelos de classificação apresentamos as métricas de desempenho utilizadas, sendo: Precisão (P), Revocação (R) e *F1-Score* (F1). Para isto, as seguintes medidas são dadas:

> **Verdadeiro positivo** (*true positive* - TP): amostras classificadas corretamente diante sua classe. Por exemplo, uma amostra de *malware* A foi classificada como A.
>
> **Falso positivo** (*false positive* — FP): amostras classificadas incorretamente diante sua classe. Por exemplo, uma amostra U foi classificada como *malware*, enquanto na verdade era benigna, representado por G.
>
> **Verdadeiro negativo** (*true negative* - TN): amostras classificadas corretamente não pertencente à outra classe. Por exemplo, uma amostra não era *malware* e o modelo previu corretamente que não era.
>
> **Falso negativo** (*false negative* — FN): amostras classificadas incorretamente não pertencente à outra classe. Por exemplo, uma amostra é um *malware* e o modelo previu como benigna.

Neste sentido, temos as seguintes definições para as métricas de desempenho adotadas por este trabalho:

$$P = \frac{TP}{TP + FP}, \tag{5.1}$$

que corresponde a taxa de observações positivas preditas corretamente sobre o total de observações positivas preditas. No âmbito de classificação de amostras maliciosas, ter baixo FP é um fator importante (Thamsirarak et al., 2015) e esta métrica torna-se relevante neste aspecto por considerar FP mais prejudicial que FN. Em sequência, temos

$$R = \frac{TP}{TP + FN}, \tag{5.2}$$

que corresponde a razão entre as observações positivas preditas corretamente e todas as observações da classe. Por fim,

$$F1 = \frac{2 \times P \times R}{P + R}, \tag{5.3}$$

que é a média harmônica entre revocação e precisão.

5.2 Resultados

Nesta seção apresentamos o primeiro experimento conduzido, onde a implementação dos *baselines* é conduzida sob a base de dados utilizada neste trabalho, assim como a aplicação do nosso método proposto para determinar se oferece resultados promissores ao compararmos com trabalhos da literatura.

5.2.1 Detecção de Android *botnet*

A Figura 5.2 mostra uma matriz, conhecida como matriz de confusão. A mesma encontra-se normalizada e dispõe os resultados obtidos por meio do método desenvolvido por Şahın et al. (2018) e aplicado na base de dados utilizada por este trabalho, usando o algoritmo KNN. Os rótulos das linhas indicam a classe real (*ground truth*) e os rótulos das colunas indicam a classe atribuída pelo algoritmo. A intensidade das cores aplicadas sobre cada célula da matriz indica o quão bem foi o percentual de acerto do algoritmo para cada classe, sendo azul escuro para representar a maior taxa de acerto, que pode chegar até 1,0, e branco para representar a menor taxa, que pode chegar até 0,0.

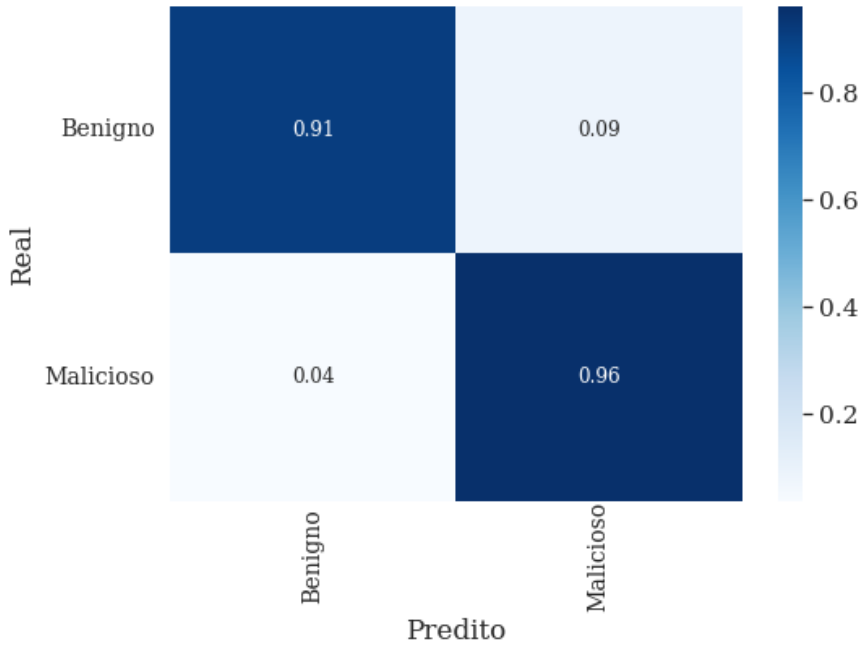


Figura 5.2: Resultado da classificação do método proposto por Şahın et al. (2018).

A matriz de confusão (Figura 5.2) mostra de forma geral que o método proposto por Şahın et al. (2018) é parcialmente bem sucedido na classificação das amostras benignas e de Android *botnet*. Ao implementarmos, a taxa de acerto ao classificar as amostras maliciosas foi de 96%, enquanto para as amostras benignas o resultado foi de 91%, sendo a acurácia geral de 94,33%. Em contrapartida, os resultados obtidos pelos autores foi de 92,43%, cerca de 1,9% a menos. Além disso, o método proposto pelos autores apresentou 8,6% de falso positivo.

Considerando um cenário real, a taxa de 8,6% de falso positivo é expressiva e isso ocorre principalmente devido a quantidade de permissões comuns entre as aplicações, dado que a categoria do conjunto de características é a mesma, o que induz o classificador ao erro. Por fim, outro fator de influência é que apenas considerar uma matriz esparsa pode oferecer um menor desempenho na classificação quando comparado à utilização da seleção de características relevantes por meio de TF-IDF, conforme será apresentado na Seção 5.2.1.1.

Em sequência, implementamos o *framework* proposto por Kirubavathi & Anitha (2018). E apesar da alta taxa de classificação obtida pelos autores, o *framework* apresenta limitações para discriminar comportamentos maliciosos e benignos baseado no conjunto de características escolhidos quando aplicado em outras bases de dados.

A Figura 5.3 mostra que o *framework* tem um baixo desempenho ao considerar `uses-feature` como conjunto de características, fazendo com que a taxa de acerto

na classificação de amostras benignas seja apenas de 72%, enquanto a classificação de Android *botnet* seja apenas de 87% e a acurácia total de 81,33%. Vimos que o `uses-feature` possui apenas 0.87% de representatividade em toda a base, enquanto `intent-filter` possui 27.99%. Essa baixa representatividade pode ser um fator limitante ao tentar extrair essas informações e, consequentemente, gerar dependência de tais características para classificar as amostras. Ao contrário disto, `intent-filter` estão associados ao componentes da aplicação, que estão presentes em qualquer aplicação. Desta forma, foi visto que a utilização de `uses-feature` influenciou negativamente nos resultados.

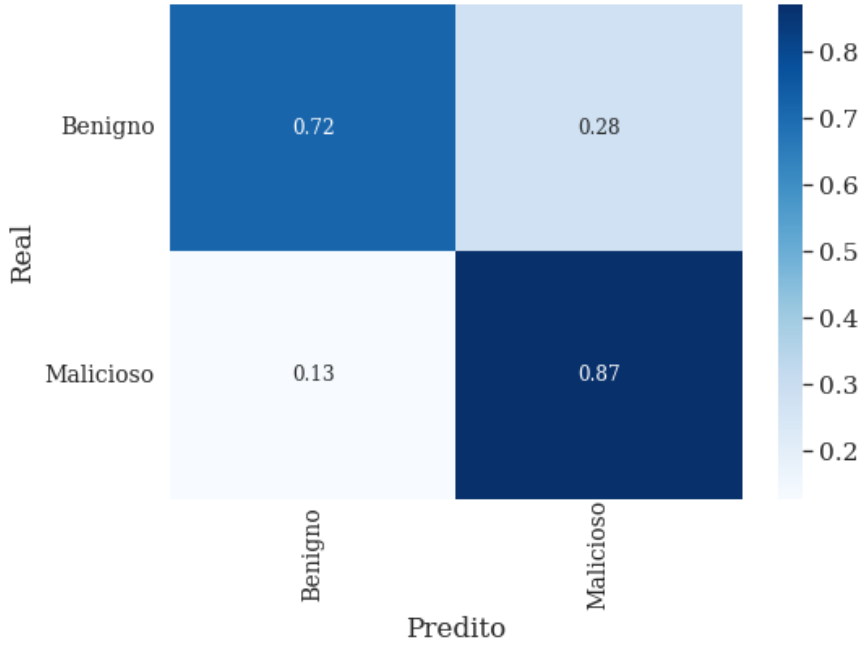


Figura 5.3: Resultado da classificação do método proposto por Kirubavathi & Anitha (2018).

Por fim, implementamos o método proposto por Hojjatinia et al. (2020) para detectar Android *botnets* por meio de Redes Neurais Convolucionais (CNN). A Figura 5.4 apresenta o resultado obtido do método proposto pelos autores utilizando a base de dados utilizada neste trabalho. Diferentemente dos demais trabalhos implementados anteriormente, este apresentou resultados promissores ao classificar as amostras, sendo 98% de acurácia na classificação das amostras benignas e 96% para classificação de amostras maliciosas. Apesar do resultado promissor, é possível observar que a utilização de um segundo conjunto de características pode ser determinante para obter de melhores resultados na taxa de detecção, assim como uma menor taxa de falso positivo, conforme propomos neste trabalho.

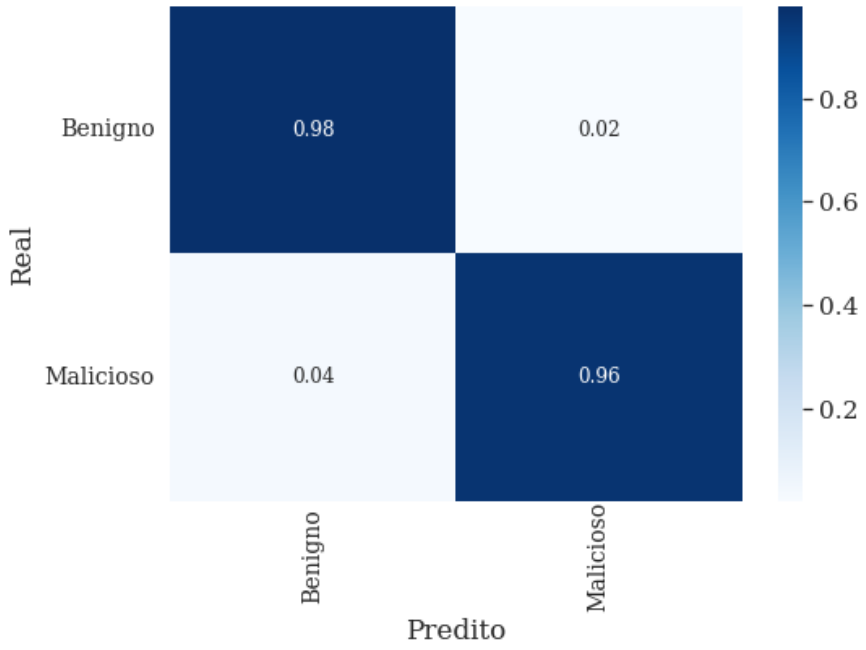


Figura 5.4: Resultado da classificação do método proposto por Hojjatinia et al. (2020).

5.2.1.1 Resultados do nosso método proposto na detecção de Android *botnet*

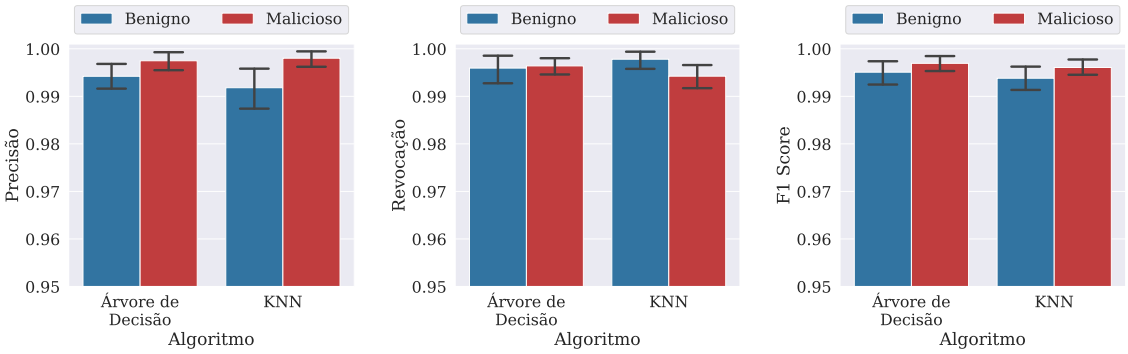
.

Nesta seção, apresentamos os resultados dos experimentos para avaliar o nosso método proposto no cenário de detecção de Android *botnet*. Os experimentos foram realizados de acordo com os critérios apresentados na Seção 5.1. O objetivo é determinar se a combinação dos conjuntos mais relevantes de características extraídas das permissões solicitadas (`uses-permission`) e dos filtros de intenção (`intent-filter`) podem fornecer resultados promissores na detecção de aplicações benignas e Android *botnet*. Com isso, nós analisamos o desempenho do método proposto usando diferentes algoritmos como Árvore de Decisão e KNN. Além disso, utilizamos os hiperparâmetros do experimento I, conforme apresentado na Tabela 5.1.

Tendo os melhores hiperparâmetros selecionados, avaliamos o desempenho de todos os algoritmos de classificação por meio de validação cruzada com 10 partições para descobrir o classificador de melhor desempenho. Em sequência, realizamos experimentos combinando os conjuntos mais relevantes de características extraídas das permissões solicitadas (`uses-permission`) e dos filtros de intenção (`intent-filter`).

Para avaliar a eficácia do nosso método proposto, utilizamos métricas como precisão, revocação, F1 e acurácia (Seção 5.1.4). Em relação à acurácia, obtivemos a taxa de acerto de 99,88% utilizando a Árvore de Decisão e 99,33% com o KNN. A Figura

5.5 apresenta o comparativo dos resultados entre os algoritmos considerando precisão, revocação e F1, enquanto a Tabela 5.4 apresenta um comparativo com os *baselines*, utilizando acurácia e F1 como métricas de avaliação.



(a) Precisão (b) Revocação (c) F1

Figura 5.5: Precisão, revocação e F1 da detecção de aplicativos maliciosos e benignos.

Podemos verificar que a acurácia é próxima entre a Árvore de Decisão e o KNN e que ambos os algoritmos obtiveram maior taxa de acerto para as aplicações maliciosas, semelhante aos *baselines*. Entretanto, diferente dos *baselines*, é possível observar uma proximidade maior entre os valores de F1 para cada classe nos algoritmos utilizados, além de uma variação menor no intervalo de confiança, o que demonstra a estabilidade dos modelos na classificação das aplicações. Em relação a este aspecto, os *baselines* apresentaram taxas menores de F1 para a detecção de aplicações benignas em relação às aplicações maliciosas.

Tabela 5.4: Comparação entre resultados do método proposto e os *baselines*.

Trabalhos	Conjunto	Acurácia	F1 (Malicioso)	F1 (Benigno)
Şahın et al. (2018)	Permissões	94,33%	0,9528±0,0160	0,9225±0,0286
Kirubavathi & Anitha (2018)	Perm. e Recursos de HW	81,33%	0,8587±0,0194	0,7546±0,0341
Hojjatinia et al. (2020)	Permissões	96,89%	0,9650±0,0157	0,9445±0,0274
Método + Árv. de decisão	Perm. e Filtros de Int.	99,88%	0,9969±0,0036	0,9951±0,0058
Método + KNN	Perm. e Filtros de Int.	99,33%	0,9961±0,0039	0,9938±0,0061

É possível observar que ao compararmos todos os resultados por meio dos dois conjuntos de características mais relevantes (`uses-permission` e `intent-filter`), o método proposto apresenta melhores resultados na classificação das amostras benignas e de Android *botnet*. O melhor resultado obtido foi por meio do algoritmo Árvore de

Decisão, com acurácia 99,88% e F1 com taxa de 0,9969 para aplicações maliciosas e 0,9951 para aplicações benignas.

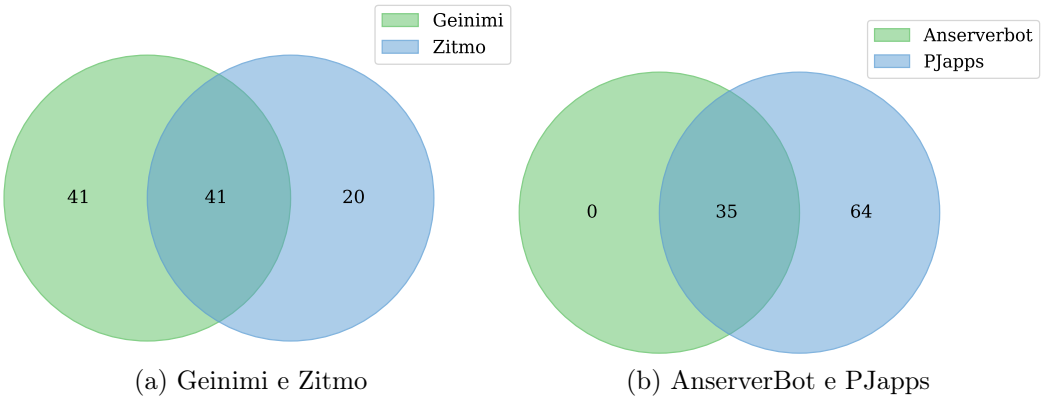
Além disso, é possível observar também que em todos os casos foi possível obter melhores resultados ao classificar amostras maliciosas ao invés de benignas. Isso ocorre devido o excesso de permissões que aplicações maliciosas costumam ter (Sokolova et al., 2017), assim como devido os filtros de intenção, o qual oferece uma semântica rica para caracterizar *malwares* (Feizollah et al., 2017), e assim fornecer maior eficácia no processo de detecção.

5.2.2 Classificação de famílias de Android *botnet*

Nesta seção apresentamos os resultados do nosso método proposto no cenário de classificação de famílias de Android *botnet*. Para isto, utilizamos a base de dados de amostras maliciosas composta por diversos tipos de *malwares* diferentes. Como parte de nossa proposta, caracterizamos e comparamos famílias de Android *botnet* para investigamos como os diferentes conjuntos de características (`uses-permission` e `intent-filter`) poderiam contribuir para o propósito deste trabalho, além do processo de detecção já apresentado anteriormente.

5.2.2.1 Experimentos sobre os diferentes conjuntos de características

Nesta seção nós conduzimos os experimentos para avaliar como os diferentes conjuntos de características (`uses-permission` e `intent-filter`) contribuem para a efetividade da classificação das famílias de Android *botnet*. A seguir, nós apresentamos a caracterização de 04 famílias. A Figura 5.6 exibe uma representação dos conjuntos de permissões utilizados por diferentes famílias de Android *botnet*.



(a) Geinimi e Zitmo (b) AnserverBot e PJapps

Figura 5.6: Comparativo entre conjunto de permissões solicitadas de famílias de Android *botnet*.

Conforme apresentado na Figura 5.6a, as famílias Geinimi e Zitmo possuem 41 permissões em comum como `READ_SMS`, presente em 166 amostras da família Geinimi (62,88%) e em 45 amostras da família Zitmo (56,25%), e `SEND_SMS`, presente em 246 amostras da família Geinimi (93,18%) e 65 amostras da família Zitmo (81,25%). Permissões relacionadas à SMS são frequentemente usadas em aplicações maliciosas, principalmente as que possuem intenções de ganho financeiro como *malwares* bancários (Seo et al., 2014), para ler e enviar as informações obtidas a partir do dispositivo comprometido.

Além disso, é possível destacar permissões que diferenciam estas famílias como a permissão `READ_HISTORY_BOOKMARKS`, que está presente em 140 amostras da família Geinimi (53,03%) e não aparece em nenhuma amostra da família Zitmo. Por outro lado, a permissão `BROADCAST_WAP_PUSH` está presente em 29 amostras da família Zitmo (36,25%) e em nenhuma amostra da família Geinimi. A permissão `READ_HISTORY_BOOKMARKS` possibilita que o aplicativo leia propriedades do navegador do dispositivo, enquanto `BROADCAST_WAP_PUSH` permite que um aplicativo emita notificações de recebimento de mensagens, tornando possível que aplicações maliciosas usem este tipo de permissão para forjar o recebimento de mensagens MMS (*Multimedia Messaging Service*).

A análise das famílias AnserverBot e PJapps (Figura 5.6b) mostra que, embora todas as permissões usadas na família AnserveBot estejam presentes na família PJapps, é possível diferenciá-las pelas permissões incomuns. Por exemplo, a permissão `READ_HISTORY_BOOKMARKS` está presente em 155 amostras da família PJapps (63,52%) e não aparece em nenhuma amostra da família AnserverBot. Outro aspecto que pode auxiliar na distinção dessas famílias é observar a frequência que as permissões em comum ocorrem. Por exemplo, a permissão `RECEIVE_BOOT_COMPLETED` encontra-se presente em 242 amostras da família AnserverBot (99,59%) e em 84 amostras da família PJapps (34,43%), e a permissão `RECEIVE_SMS`, está presente em 242 amostras da família AnserverBot (99,59%) e 164 amostras da família PJapps (67,21%).

Essas similaridades de uso de permissões apresentadas em algumas famílias de Android *botnet* torna o processo de detecção ou de classificação ainda mais desafiador. Por essa razão, alguns trabalhos argumentam que apenas a utilização das permissões pode não ser o suficiente para caracterizar comportamentos maliciosos (Aafer et al., 2013).

Para lidar com esse problema, este trabalho propõe a inclusão de outro conjunto de características extraído a partir das ações definidas entre os componentes da aplicação por meio de filtros de intenção. A Figura 5.7 exibe a relação entre as ações comuns e distintas executadas pelas diferentes famílias de Android *botnet*.

A Figura 5.7 mostra que os filtros de intenção podem fornecer características mais precisas das relações entre as famílias de Android *botnet*. Por exemplo, o filtro de intenção `CONNECTIVITY_CHANGE` está presente em 14 amostras da família Geinimi (5,30%) e em nenhuma amostra da família Zitmo. Por outro lado, o filtro de intenção `REBOOT` está presente em 16 amostras da família Zitmo (16,25%) e em nenhuma amostra da família Geinimi. Essa relação de conjuntos disjuntos também pode ser observada nas demais famílias. Isto significa dizer que um número maior de características únicas pode resultar em uma melhor distinção entre as famílias. A Tabela 5.5 apresenta as 10 características mais relevantes (05 permissões e 05 filtros de intenção) das 04 famílias de Android *botnet*.

(a) Geinimi e Zitmo (b) AnserverBot e PJapps

Figura 5.7: Comparativo entre conjunto de filtros de intenção de famílias de Android *botnet*.

Os resultados da Tabela 5.5 mostram que um grande percentual das operações dessas famílias de Android *botnet* são derivadas das permissões de leitura ou escrita de dados sensitivos, e que as ações executadas por meio de filtros de intenção tipicamente exploram o envio e o recebimento de mensagens. Por exemplo, a família PJapps possui permissões como `READ_HISTORY_BOOKMARKS` e `WRITE_HISTORY_BOOKMARKS`, que permitem a leitura e manipulação das propriedades do navegador do dispositivo do usuário, e filtros de intenção como `SIG_STR` e `SMS_RECEIVED` que são responsáveis por acionar comportamentos maliciosos a partir de um serviço iniciado pelo próprio *malware* (Lin et al., 2016) e notificar o recebimento de mensagens SMS para obter informações sensíveis (Chen et al., 2016).

Tabela 5.5: Características de maior relevância por família.

Família	Permissão	TF-IDF	Filtro de Intenção	TF-IDF
Geinimi	ACCESS_LOCATION	0.321643	BOOT_COMPLETED	0.696994
	ACCESS_GPS	0.321643	QUICKBOOT_POWERON	0.364238
	SEND_SMS	0.276381	IEXTENDEDNETWORKSERVICE	0.364238
	READ_HISTORY_BOOKMARKS	0.261642	SMS_RECEIVED	0.280386
	WRITE_HISTORY_BOOKMARKS	0.261642	PHONE_STATE	0.202927
Zitmo	READ_PHONE_STATE	0.263232	SMS_RECEIVED	0.561762
	RECEIVE_SMS	0.230788	APPWIDGET_UPDATE	0.472411
	BROADCAST_WAP_PUSH	0.223447	BOOT_COMPLETED	0.410933
	WRITE_SECURE	0.223447	ACCESSIBILITYSERVICE	0.218633
	CHANGE_WIFI_STATE	0.221941	NEW_OUTGOING_CALL	0.205853
AnserverBot	RESTART_PACKAGES	0.268175	PICK_WIFI_WORK	0.456310
	READ_LOGS	0.267081	UMS_CONNECTED	0.394942
	WRITE_CONTACTS	0.264892	UMS_DISCONNECTED	0.394942
	WRITE_APN_SETTINGS	0.263797	MEDIA_NOFS	0.393296
	READ_CONTACTS	0.249343	INPUT_METHOD_CHANGED	0.351401
PJapps	READ_HISTORY_BOOKMARKS	0.368279	SIG_STR	0.729103
	WRITE_HISTORY_BOOKMARKS	0.365903	SMS_RECEIVED	0.370022
	INTERNET	0.353369	BOOT_COMPLETED	0.236341
	READ_PHONE_STATE	0.336011	SEND	0.230465
	WRITE_EXTERNAL_STORAGE	0.264097	PHONE_STATE	0.195076

Na Tabela 5.5, outra observação importante é que o peso de cada característica atribuído pelo TF-IDF serve não somente para determinar a relevância da característica como também pode ser usado para diferenciar famílias com características semelhantes, como é o caso das famílias Geinimi e PJapps. Por exemplo, a permissão `READ_HISTORY_BOOKMARKS` possui o peso do TF-IDF de 0.261642 na família Geinimi, enquanto na família PJapps a mesma permissão possui o peso 0.368279. Além disso, o conjunto e a ordem dos filtros de intenção em comum também se diferenciam entre estas famílias, o que confirma a importância da abordagem empregada.

Para mais detalhes, o Anexo A apresenta uma tabela com a lista de todas as características de todas as famílias, incluindo as já apresentadas anteriormente.

5.2.2.2 Resultados do nosso método proposto na classificação de famílias de Android *botnet*

Nesta seção, nós analisamos o desempenho do nosso método proposto no cenário de classificação de famílias de Android *botnet*, visto que a combinação dos conjuntos mais relevantes de características extraídas das permissões solicitadas (`uses-permission`) e dos filtros de intenção (`intent-filter`) podem fornecer conhecimento adicional na caracterização dos comportamentos maliciosos destas famílias (Seção 5.2.2.1). Os experimentos foram realizados de acordo com os critérios definidos na Seção 5.1, assim como a utilização dos hiperparâmetros do experimento II, apresentados na Tabela 5.1. A Figura 5.8 e a Tabela 5.6 apresentam um comparativo dos resultados obtidos por meio dos algoritmos de Árvore de Decisão e KNN.

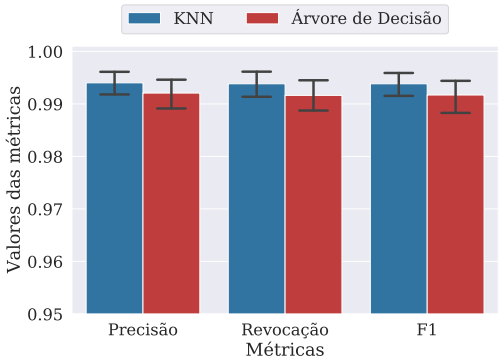


Figura 5.8: Precisão, revocação e F1 da classificação de famílias de Android *botnet*.

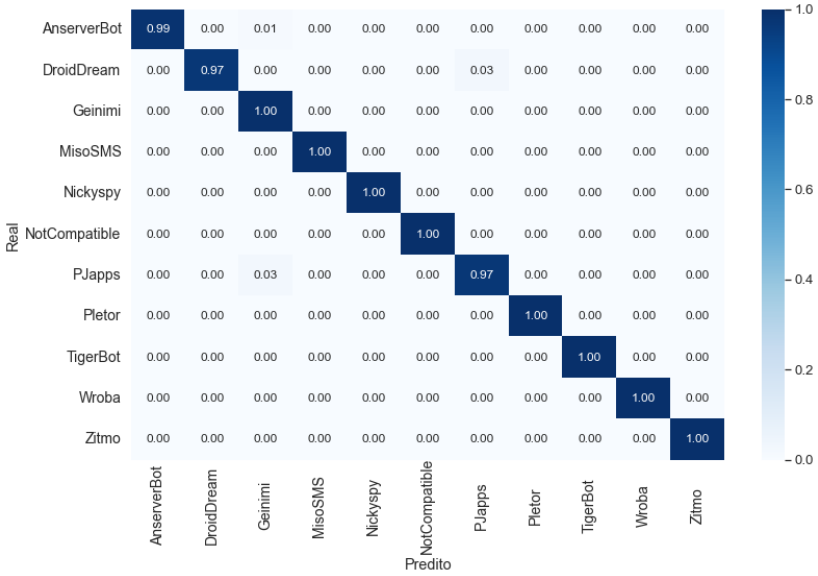
A Árvore de Decisão obteve acurácia de 99,29% e o F1 de 0,9917, enquanto o KNN, com melhor desempenho, obteve acurácia de 99,38% e o F1 de 0,9938 para a classificação de famílias de Android *botnet*. Neste contexto, verificamos que os resultados para o método proposto com ambos os algoritmos manteve a estabilidade na classificação, mesmo com a base de dados com classes desproporcionais.

Tabela 5.6: Comparação entre resultados do método proposto para a classificação de famílias de Android *botnet*.

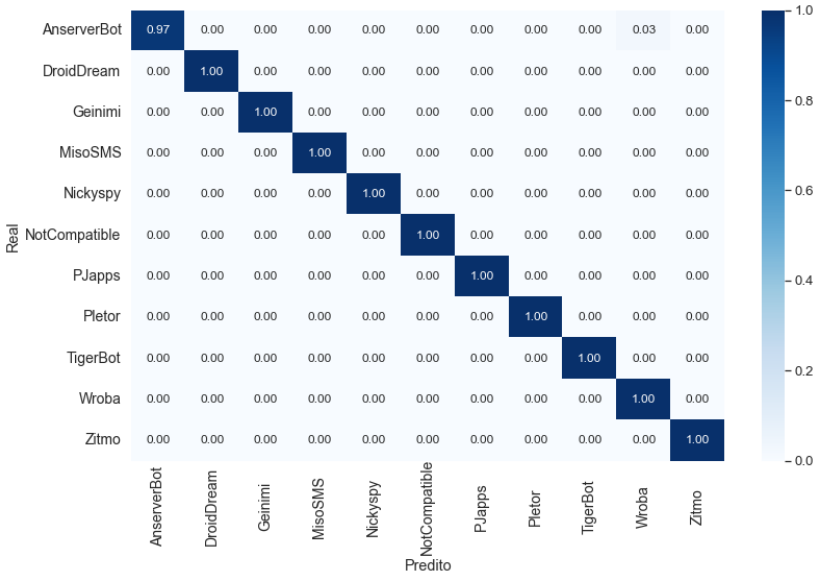
Abordagem	Acurácia	Precisão	Revocação	F1
Método + Árv. de decisão	99,29%	0,9921±0,0069	0,9916±0,0073	0,9917±0,0073
Método + KNN	99,38%	0,9940±0,0053	0,9938±0,0055	0,9938±0,0055

Para auxiliar na compreensão dos resultados, a Figura 5.9 mostra uma matriz de confusão para os resultados do KNN e da Árvore de Decisão. Podemos verificar

que o método proposto com ambos os algoritmos não obteve 100% de taxa de acerto apenas ao classificar a famílias AnserverBot, além das famílias DroidDream e PJapps no caso do KNN. Para a Árvore de Decisão, 3% das amostras da família AnserverBot foram classificadas como Wroba. Para o KNN, 1% das amostras de AnserverBot foram classificadas como Geinimi, assim como 3% das amostras de DroidDream foram classificadas como PJapps. Estas, por sua vez, tiveram 3% das amostras classificadas como Geinimi.



(a) KNN



(b) Árvore de Decisão

Figura 5.9: Matriz de confusão dos resultados obtidos para a classificação de famílias de Android *botnet*.

A classificação incorreta pode ser devido à natureza das famílias, dado que possuem o mesmo tipo de C&C e ataques em comum, o que pode influenciar no conjunto de características. Além disso, na Tabela 5.5 podemos verificar, por exemplo, semelhanças entre as permissões e os filtros de intenção considerados de maior importância entre as famílias PJapps e Geinimi. Dentre as permissões, temos `READ_HISTORY_BOOKMARKS` e `WRITE_HISTORY_BOOKMARKS`, enquanto para os filtros de intenção temos `SMS_RECEIVED` e `PHONE_STATE`. Essas famílias possuem HTTP como tipo de C&C e *repackaged application* como ataque em comum, o que influencia no resultado obtido.

Estes resultados indicam a existência de comportamentos compartilhados entre famílias, induzindo os classificadores a erro. Entretanto, esse achado também permite pensar em soluções que possam ser reaproveitadas entre amostras que apresentam comportamentos similares.

5.3 Considerações Finais

Neste capítulo, exploramos diferentes conjuntos de características e algoritmos de aprendizagem de máquina para realizar a detecção de Android *botnet*, assim como a classificação de famílias.

Os algoritmos utilizados neste trabalho foram o KNN e a Árvore de Decisão por meio do método proposto. Além disso, aplicamos o *Grid Search* para ajustarmos os hiperparâmetros dos classificadores, conforme apresentado na Tabela 5.1. Ao analisarmos esta tabela, verificamos os seguintes aspectos em relação aos algoritmos:

- Árvore de Decisão: utilizamos a entropia para medir a qualidade da divisão entre os nós da árvore, além do parâmetro de profundidade *max_ depth* para regularizar a árvore ou limitar a maneira como ela cresce para evitar o ajuste excessivo.
- KNN: utilizamos os vizinhos mais próximos considerando a métrica de distância *manhattan*. Dentre os vizinhos selecionados, os mais próximos do ponto tiveram uma influência maior do que os vizinhos mais distantes para definir o rótulo.

Com estes hiperparâmetros aplicados ao primeiro experimento, detecção de Android *botnet*, o KNN obteve acurácia de 99,33% e F1 com taxa de 0,9961 para aplicações maliciosas e 0,9938 para aplicações benignas, enquanto a Árvore de Decisão, com melhor desempenho, obteve acurácia de 99,88% e F1 com taxa de 0,9969 para aplicações maliciosas e 0,9951 para aplicações benignas. Os resultados em ambos os algoritmos utilizando o método proposto tiveram resultados acima dos *baselines* apresentados

neste capítulo. Considerando o segundo experimento, classificação de famílias de Android *botnet*, o KNN obteve acurácia de 99,38% e F1 com taxa de 0,9938, enquanto a Árvore de Decisão obteve acurácia de 99,29% e F1 com taxa de 0,9917 ao classificar as famílias de Android *botnet*.

Tais resultados mostram que o nosso método proposto pode ser considerado para diferentes cenários como detecção de Android *botnet* e classificação de famílias desse tipo de ameaça. Neste caso, a utilização do TF-IDF possibilitou que apenas as permissões e os filtros de intenção mais relevantes fossem utilizados, diminuindo o conjunto de características e mantendo a estabilidade na classificação, mesmo diante de classes com quantidades de amostras desproporcionais.

Para chegar em tais resultados, foram realizados experimentos quantitativos, o que nos permitiram uma análise exploratória dos conjuntos de características extraídos das 11 famílias de Android *botnet* do mundo real. Por meio destes experimentos, mostramos que a adição dos filtros de intenção contribuiu para a melhor caracterização destas famílias, visto que um grande percentual das operações são derivadas de leitura ou escrita de dados sensitivos estão relacionadas com o tipo de C&C e as características de ataque.

No Capítulo 6 apresentamos as conclusões obtidas por meio do estudo realizado assim como as limitações descobertas. Por fim, apresentamos os trabalhos futuros.

Capítulo 6

Conclusões

Neste trabalho, apresentamos um método de detecção de Android *botnet* baseado na relevância de permissões e filtros de intenção. Por meio das diferentes estratégias de avaliação presentes nos experimentos, mostramos que o método proposto distingue com precisão aplicativos benignos e maliciosos, assim como as famílias de Android *botnet*, mostrando que a eficácia do método proposto pode ser empregado para cenários diferentes.

Entre as descobertas realizadas neste trabalho, destacamos que nossa abordagem pode classificar com precisão as aplicações benignas e maliciosas usando apenas quantificadores de recuperação da informação para definir os pesos das características de cada classe e assim atingir valores de acurácia e F1 mais altos, mesmo usando apenas algoritmos de aprendizado de máquina tradicionais. Essa descoberta mostra que a aplicação de quantificadores de recuperação da informação para pontuar matrizes de termos podem melhorar os resultados de eficácia de detecção de Android *botnet* por meio de algoritmos de aprendizado de máquina.

Além disso, em comparação com os *baselines*, nosso método apresentou ganhos de até 18,55% na acurácia, e para o F1, de até 13,82%. O ganho na eficácia é resultado do conjunto de características que usamos, assim como devido a redução de dimensionalidade, representados pelas permissões e filtros de intenção, presentes em todas as amostras. Além disso, destacamos que a utilização de tais conjuntos de características podem ser utilizadas não apenas para detecção, mas também em problemas de classificação de famílias..

Por fim, a construção de modelos de aprendizagem com recursos extraídos de permissões e filtros de intenção pode levar a resultados de eficácia competitiva, quando comparados com modelos construídos com base em permissões ou recursos de baixa representatividade no aplicativos. Apenas considerar um conjunto de características

pode induzir os classificadores ao erro se houver alta similaridade ou baixa representatividade. Desta forma, tornando-se importante adotar mais de um conjunto, preferencialmente o que seja inerente às aplicações e arquitetura.

6.1 Contribuições

As principais contribuições deste trabalho são:

1. Demonstração da eficácia de dois conjuntos de características obtidos do arquivo de configuração do Android (*AndroidManifest.xml*) para auxiliar na detecção de Android *botnet*. Especificamente, nós observamos as permissões solicitadas ao usuário (definida como `uses-permission`) e as ações à serem realizadas pelos componentes da aplicação por meio de filtros de intenção (definida como `intent-filter`);

2. Um método de detecção de Android *botnet* por meio de aplicações do mundo real, empregando uma medida de avaliação presente no campo de processamento de recuperação da informação e algoritmos de aprendizado de máquina.

3. Uma análise do método proposto para classificação de famílias de *malwares*.

6.2 Limitações

Como limitações deste trabalho, podemos destacar a necessidade em encontrar conjuntos com dados representativos diante um grande número de amostras. Também podemos mencionar que, como nossa abordagem é baseada em análise estática, provavelmente não funcionará bem em um cenário onde haja poucas amostras, já que dificultaria o processo de aprendizado do algoritmo, considerando apenas os dois conjuntos de amostras adotados. Além disso, outra limitação é que apenas análise estática pode limitar o entendimento diante cenários de ataque onde há exfiltração de dados ou comunicação com C&C, sendo assim necessário utilizar técnicas como análise dinâmica para ter melhor compreensão da ameaça em questão.

6.3 Trabalhos Futuros

Como trabalhos futuros ou variações deste trabalho podem ser elencados:

- Estudos de novos conjuntos de características para identificar quais podem maximizar os resultados para conjuntos de amostras com quantidade inferior à 50 aplicativos;
- Aplicar algoritmos de classificação hierárquica para validar a eficácia do método proposto diante diferentes classes;
- Expandir a base de dados e aplicar o método diante novos conjuntos de amostras maliciosas, dado que novas famílias de *malwares* surgem diariamente;
- Combinar a técnica de análise estática com a técnica de análise dinâmica para melhorar a detecção da superfície de ataque.

Referências Bibliográficas

Aafer, Y.; Du, W. & Yin, H. (2013). Droidapiminer: Mining api-level features for robust malware detection in android. Em Zia, T.; Zomaya, A.; Varadharajan, V. & Mao, M., editores, *Security and Privacy in Communication Networks*, pp. 86--103, Cham. Springer International Publishing.

Abdul Kadir, A. F.; Stakhanova, N. & Ghorbani, A. A. (2015). Android botnets: What urls are telling us. Em Qiu, M.; Xu, S.; Yung, M. & Zhang, H., editores, *Network and System Security*, pp. 78--91, Cham. Springer International Publishing.

Alqatawna, J.; Ala'M, A.-Z.; Hassonah, M. A.; Faris, H. et al. (2021). Android botnet detection using machine learning models based on a comprehensive static analysis approach. *Journal of Information Security and Applications*, 58:102735.

Android (2020a). App Manifest Overview). https://developer.android.com/guide/topics/manifest/manifest-intro. [Online; acessado 01-Agosto-2020].

Android (2020b). Application Fundamentals). https://developer.android.com/guide/components/fundamentals. [Online; acessado 01-Agosto-2020].

Android (2022a). Android Open Source Project. https://source.android.com/. [Online; acessado 24-Abril-2022].

Android (2022b). Platform Architecture. https://developer.android.com/guide/platform. [Online; acessado 24-Abril-2022].

Anwar, S.; Zain, J. M.; Inayat, Z.; Haq, R. U.; Karim, A. & Jabir, A. N. (2016). A static approach towards mobile botnet detection. Em *2016 3rd International Conference on Electronic Design (ICED)*, pp. 563–567. ISSN null.

Aresu, M.; Ariu, D.; Ahmadi, M.; Maiorca, D. & Giacinto, G. (2015). Clustering android malware families by http traffic. Em *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 128–135.

Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K. & Siemens, C. (2014). Drebin: Effective and explainable detection of android malware in your pocket. Em *NDSS*, volume 14, pp. 23--26.

Bergman, N.; Rouse, J.; Stanfield, M.; Scambray, J.; Deshmukh, S.; Price, M.; Geethakumar, S.; Matsumoto, S. & Steven, J. (2013). *Hacking Exposed Mobile: Security Secrets & Solutions*. McGraw Hill Professional.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer Science+ Business Media.

Chen, W.; Aspinall, D.; Gordon, A. D.; Sutton, C. & Muttik, I. (2016). More semantics more robust: Improving android malware classifiers. Em *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, p. 147–158, New York, NY, USA. Association for Computing Machinery.

da Costa, V. G. T.; Barbon, S.; Miani, R. S.; Rodrigues, J. J. P. C. & Zarpelão, B. B. (2017). Detecting mobile botnets through machine learning and system calls analysis. Em *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6.

da Silva, L. A.; Peres, S. M. & Boscarioli, C. (2017). *Introdução à mineração de dados: com aplicações em R*. Elsevier Brasil.

Dagon, D.; Zou, C. C. & Lee, W. (2006). Modeling botnet propagation using time zones. Em *NDSS*, volume 6, pp. 2--13.

Derakhshan, F. & Ashrafnejad, M. (2020). *The Risk of Botnets in Cyber Physical Systems*, pp. 81--106. Springer International Publishing, Cham.

Faceli, K.; Lorena, A. C.; Gama, J.; Carvalho, A. C. P. d. L. et al. (2011). *Inteligência Artificial: Uma abordagem de aprendizado de máquina*. Rio de Janeiro: LTC.

Farina, P.; Cambiaso, E.; Papaleo, G. & Aiello, M. (2016). Are mobile botnets a possible threat? the case of slowbot net. *Computers & Security*, 58:268 – 283. ISSN 0167-4048.

Faruki, P.; Bharmal, A.; Laxmi, V.; Ganmoor, V.; Gaur, M. S.; Conti, M. & Rajarajan, M. (2014). Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 17(2):998--1022.

Feizollah, A.; Anuar, N. B.; Salleh, R.; Suarez-Tangil, G. & Furnell, S. (2017). Andro-dialysis: Analysis of android intent effectiveness in malware detection. volume 65, pp. 121 – 134. ISSN 0167-4048.

Geisser, S. (1974). A predictive approach to the random effect model. *Biometrika*, 61(1):101--107.

Gezer, A.; Warner, G.; Wilson, C. & Shrestha, P. (2019). A flow-based approach for trickbot banking trojan detection. *Computers Security*, 84:179 – 192. ISSN 0167-4048.

Goldschmidt, R.; Bezerra, E. & Passos, E. (2015). *Data Mining: Conceitos, técnicas, algoritmos, orientações e aplicações*. Elsevier.

González, G.; Lárraga, M. E.; Alvarez-Icaza, L. & Gomez, J. (2021). Bluetooth worm propagation in smartphones: Modeling and analyzing spatio-temporal dynamics. *IEEE Access*, 9:75265–75282.

Hiemstra, D. (2000). A probabilistic justification for using tf× idf term weighting in information retrieval. *International Journal on Digital Libraries*, 3(2):131--139.

Hijawi, W.; Alqatawna, J. & Faris, H. (2017). Toward a detection framework for android botnet. Em *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, pp. 197–202. ISSN null.

Hojjatinia, S.; Hamzenejadi, S. & Mohseni, H. (2020). Android botnet detection using convolutional neural networks. Em *2020 28th Iranian Conference on Electrical Engineering (ICEE)*, pp. 1--6. IEEE.

Huang, C.-Y.; Tsai, Y.-T. & Hsu, C.-H. (2013). Performance evaluation on permission-based detection for android malware. Em Pan, J.-S.; Yang, C.-N. & Lin, C.-C., editores, *Advances in Intelligent Systems and Applications - Volume 2*, pp. 111--120, Berlin, Heidelberg. Springer Berlin Heidelberg.

IDC (2022). Smartphone Market Share. https://www.idc.com/promo/smartphone-market-share/os. [Online; acessado 24-Abril-2022].

Jang, J.-w.; Yun, J.; Woo, J. & Kim, H. K. (2014). Andro-profiler: Anti-malware system based on behavior profiling of mobile malware. Em *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, p. 737–738, New York, NY, USA. Association for Computing Machinery.

Karim, A.; Salleh, R. & Shah, S. A. A. (2015). Dedroid: a mobile botnet detection approach based on static analysis. Em *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pp. 1327--1332. IEEE.

Kirubavathi, G. & Anitha, R. (2018). Structural analysis and detection of android botnets using machine learning techniques. volume 17, pp. 153--167. ISSN 1615-5270.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. Em *Ijcai*, volume 14, pp. 1137--1145. Montreal, Canada.

Kothari, S. & Joshi, S. (2020). Analysis of android applications to detect botnet attacks. Em *2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC)*, pp. 144–150. IEEE.

Koyuncu, M. & Pusatli, T. (2019). Security awareness level of smartphone users: An exploratory case study. *Mobile Information Systems*, 2019.

Lecheta, R. R. (2013). *Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK*. Novatec Editora.

Lin, Z.; Wang, R.; Jia, X.; Zhang, S. & Wu, C. (2016). Analyzing android repackaged malware by decoupling their event behaviors. Em Ogawa, K. & Yoshioka, K., editores, *Advances in Information and Computer Security*, pp. 3--20, Cham. Springer International Publishing.

Marsland, S. (2014). *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Pieterse, H. & Olivier, M. S. (2012). Android botnets on the rise: Trends and characteristics. Em *2012 Information Security for South Africa*, pp. 1–5. ISSN 2330-9881.

Ramos, J. et al. (2003). Using tf-idf to determine word relevance in document queries. Em *Proceedings of the first instructional conference on machine learning*, volume 242, pp. 133--142. Piscataway, NJ.

Rovelli, P. & Vigfússon, Ý. (2014). Pmds: Permission-based malware detection system. Em Prakash, A. & Shyamasundar, R., editores, *Information Systems Security*, pp. 338--357, Cham. Springer International Publishing.

Şahın, D. Ö.; Kural, O. E.; Akleylek, S. & Kiliç, E. (2018). New results on permission based static analysis for android malware. Em *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, pp. 1--4.

Samra, A. A. A.; Kangbin Yim & Ghanem, O. A. (2013). Analysis of clustering technique in android malware detection. Em *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 729–733.

Sanz, B.; Santos, I.; Laorden, C.; Ugarte-Pedrero, X.; Bringas, P. G. & Álvarez, G. (2013). Puma: Permission usage to detect malware in android. Em Herrero, Á.; Snášel, V.; Abraham, A.; Zelinka, I.; Baruque, B.; Quintián, H.; Calvo, J. L.; Sedano, J. & Corchado, E., editores, *International Joint Conference CISIS'12-ICEUTE12-SOCO12 Special Sessions*, pp. 289--298, Berlin, Heidelberg. Springer Berlin Heidelberg.

Seo, S.-H.; Gupta, A.; Mohamed Sallam, A.; Bertino, E. & Yim, K. (2014). Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications*, 38:43 – 53. ISSN 1084-8045.

Silva, S. S.; Silva, R. M.; Pinto, R. C. & Salles, R. M. (2013). Botnets: A survey. *Computer Networks*, 57(2):378 – 403. ISSN 1389-1286. Botnet Activity: Analysis, Detection and Shutdown.

Sokolova, K.; Perez, C. & Lemercier, M. (2017). Android application classification and anomaly detection with graph-based permission patterns. *Decision Support Systems*, 93:62–76. ISSN 0167-9236.

Taheri, L.; Kadir, A. F. A. & Lashkari, A. H. (2019). Extensible android malware detection and family classification using network-flows and api-calls. Em *2019 International Carnahan Conference on Security Technology (ICCST)*, pp. 1–8. IEEE.

Talal, M.; Zaidan, A.; Zaidan, B.; Albahri, O. S.; Alsalem, M.; Albahri, A. S.; Alamoodi, A.; Kiah, M. L. M.; Jumaah, F. & Alaa, M. (2019). Comprehensive review and analysis of anti-malware apps for smartphones. *Telecommunication Systems*, 72(2):285--337.

Tata, S. & Patel, J. M. (2007). Estimating the selectivity of tf-idf based cosine similarity predicates. *SIGMOD Rec.*, 36(2):7–12. ISSN 0163-5808.

Thamsirarak, N.; Seethongchuen, T. & Ratanaworabhan, P. (2015). A case for malware that make antivirus irrelevant. Em *2015 12th International Conference on Electrical*

Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), pp. 1–6.

Wang, W.; Li, Y.; Wang, X.; Liu, J. & Zhang, X. (2018). Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. *Future Generation Computer Systems*, 78:987 – 994. ISSN 0167-739X.

Wang, W.; Wang, X.; Feng, D.; Liu, J.; Han, Z. & Zhang, X. (2014). Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9(11):1869–1882.

Wang, X.; Wang, W.; He, Y.; Liu, J.; Han, Z. & Zhang, X. (2017). Characterizing android apps' behavior for effective detection of malapps at large scale. *Future Generation Computer Systems*, 75:30 – 45. ISSN 0167-739X.

Wu, D.; Mao, C.; Wei, T.; Lee, H. & Wu, K. (2012). Droidmat: Android malware detection through manifest and api calls tracing. Em *2012 Seventh Asia Joint Conference on Information Security*, pp. 62–69.

Xiang, C.; Binxing, F.; Lihua, Y.; Xiaoyi, L. & Tianning, Z. (2011). Andbot: Towards advanced mobile botnets. Em *Proceedings of the 4th USENIX Conference on Large-Scale Exploits and Emergent Threats*, LEET'11, p. 11, USA. USENIX Association.

Xu, K.; Li, Y. & Deng, R. H. (2016). Iccdetector: Icc-based malware detection on android. *IEEE Transactions on Information Forensics and Security*, 11(6):1252–1264. ISSN 1556-6021.

Yerima, S. Y. & Alzaylaee, M. K. (2020). Mobile botnet detection: A deep learning approach using convolutional neural networks. Em *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, pp. 1–8.

Yerima, S. Y.; Alzaylaee, M. K.; Shajan, A. et al. (2021). Deep learning techniques for android botnet detection. *Electronics*, 10(4):519.

Yusof, M.; Saudi, M. M. & Ridzuan, F. (2017). A new mobile botnet classification based on permission and api calls. Em *2017 Seventh International Conference on Emerging Security Technologies (EST)*, pp. 122–127. ISSN 2472-7601.

Zhongyang, Y.; Xin, Z.; Mao, B. & Xie, L. (2013). Droidalarm: An all-sided static analysis tool for android privilege-escalation malware. Em *Proceedings of the 8th*

ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS '13, p. 353–358, New York, NY, USA. Association for Computing Machinery.

Zulkefli, Z. & Mahinderjit Singh, M. (2020). Sentient-based access control model: A mitigation technique for advanced persistent threats in smartphones. *Journal of Information Security and Applications*, 51:102431. ISSN 2214-2126.

Anexo A

Características de maior relevância para todas as famílias de Android botnet

Tabela A.1: Características de maior relevância por família.

Família	Permissão	TF-IDF	Filtro de Intenção	TF-IDF
Geinimi	ACCESS_LOCATION	0.321643	BOOT_COMPLETED	0.696994
	ACCESS_GPS	0.321643	QUICKBOOT_POWERON	0.364238
	SEND_SMS	0.276381	IEXTENDEDNETWORKSERVICE	0.364238
	READ_HISTORY_BOOKMARKS	0.261642	SMS_RECEIVED	0.280386
	WRITE_HISTORY_BOOKMARKS	0.261642	PHONE_STATE	0.202927
Zitmo	READ_PHONE_STATE	0.263232	SMS_RECEIVED	0.561762
	RECEIVE_SMS	0.230788	APPWIDGET_UPDATE	0.472411
	BROADCAST_WAP_PUSH	0.223447	BOOT_COMPLETED	0.410933
	WRITE_SECURE	0.223447	ACCESSIBILITYSERVICE	0.218633
	CHANGE_WIFI_STATE	0.221941	NEW_OUTGOING_CALL	0.205853
AnserverBot	RESTART_PACKAGES	0.268175	PICK_WIFI_WORK	0.456310
	READ_LOGS	0.267081	UMS_CONNECTED	0.394942
	WRITE_CONTACTS	0.264892	UMS_DISCONNECTED	0.394942
	WRITE_APN_SETTINGS	0.263797	MEDIA_NOFS	0.393296

Continua na próxima página

Tabela A.1 – *Continuação da página anterior*

Família	Permissão	TF-IDF	Filtro de Intenção	TF-IDF
	READ_CONTACTS	0.249343	INPUT_METHOD_CHANGED	0.351401
PJapps	READ_HISTORY_BOOKMARKS	0.368279	SIG_STR	0.729103
	WRITE_HISTORY_BOOKMARKS	0.365903	SMS_RECEIVED	0.370022
	INTERNET	0.353369	BOOT_COMPLETED	0.236341
	READ_PHONE_STATE	0.336011	SEND	0.230465
	WRITE_EXTERNAL_STORAGE	0.264097	PHONE_STATE	0.195076
NotCompatible	RECEIVE_BOOT_COMPLETED	0.578954	USER_PRESENT	0.756872
	INTERNET	0.578954	BOOT_COMPLETED	0.648693
	ACCESS_NETWORK_STATE	0.556951	SYNCRESOURCESERVICE	0.023829
	READ_INTERNAL_STORAGE	0.021194	DOWNLOADQUEUE	0.023829
	WRITE_INTERNAL_STORAGE	0.021194	USB_DISCONNECTED	0.023829
DroidDream	INTERNET	0.509942	PHONE_STATE	0.665382
	ACCESS_NETWORK_STATE	0.424662	BOOT_COMPLETED	0.58246
	READ_PHONE_STATE	0.388983	PACKAGE_ADDED	0.223361
	RECEIVE_BOOT_COMPLETED	0.261063	PACKAGE_REMOVED	0.178689
	READ_CONTACTS	0.259274	INSTALLEDREQUESTSERVICE	0.152499
Pletor	READ_EXTERNAL_STORAGE	0.356976	EXTERNAL_APPLICATIONS_AVAILABLE	0.737238
	CAMERA	0.356976	DEVICE_ADMIN_ENABLED	0.397825
	SEND_SMS	0.304976	SMS_RECEIVED	0.331831
	READ_SMS	0.301345	BOOT_COMPLETED	0.308947
	RECEIVE_SMS	0.301345	CONNECTIVITCHANGE	0.304395
MisoSMS	ADD_SYSTEM_SERVICE	0.35478	BOOT_COMPLETED	0.678366
	READ_CALL_LOG	0.291426	DEVICE_ADMIN_ENABLED	0.677423
	WRITE_CALL_LOG	0.287203	SMS_RECEIVED	0.173061
	MOUNT_UNMOUNT_FILESYSTEMS	0.263058	USER_PRESENT	0.049105
	BROADCAST_STICKY	0.249116	PACKAGE_REMOVED	0.044567
Nickyspy	INTERNET	0.375795	MEDIA_MOUNTED	0.470636
	READ_PHONE_STATE	0.35347	BOOT_COMPLETED	0.421823

Continua na próxima página

Tabela A.1 – *Continuação da página anterior*

Família	Permissão	TF-IDF	Filtro de Intenção	TF-IDF
	WRITE_EXTERNAL_STORAGE	0.325565	SMS_SENT	0.268321
	ACCESS_NETWORK_STATE	0.273474	PRECHANNEL	0.253414
	RECEIVE_BOOT_COMPLETED	0.258591	PLAYURL	0.253414
TigerBot	REBOOT	0.357212	DEVICE_ADMIN_ENABLED	0.628124
	BAIDU_LOCATION_SERVICE	0.314137	SMS_RECEIVED	0.559696
	MODIFY_PHONE_STATE	0.224407	BOOT_COMPLETED	0.520314
	PROCESS_OUTGOING_CALLS	0.205601	PHONE_STATE	0.090729
	ACCESS_COARSE_LOCATION	0.205601	NOUTGOING_CALL	0.090729
Wroba	UPDATE_APP_OPS_STATS	0.350099	BOOT_COMPLETED	0.277786
	READ_PHONE_STATE	0.272704	SMS_RECEIVED	0.068867
	VIBRATE	0.237700	BOOT_COMPLETED	0.277786
	GET_TASKS	0.232993	PHONE_STATE	0.067984
	READ_CONTACTS	0.232993	NOUTGOING_CALL	0.067984