



UNIVERSIDADE FEDERAL DE PERNAMBUCO  
CENTRO DE INFORMÁTICA  
DEPARTAMENTO DE INFORMÁTICA  
GRADUAÇÃO EM SISTEMAS DA INFORMAÇÃO

VICTOR AUGUSTO PEREIRA BURGARDT

**Análise da importância de características na detecção  
de *Malwares* para sistemas *Android***

Trabalho de Conclusão de Curso (TCC) de Graduação

Recife

2019

VICTOR AUGUSTO PEREIRA BURGARDT

**Análise da importância de características na detecção  
de *Malwares* para sistemas *Android***

Trabalho apresentado ao Programa de Graduação em Sistemas da Informação do Departamento de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas da informação.

**Área de concentração:** Ciência dos Dados

**Orientador:** Prof. Dr. *Paulo Salgado Gomes de Mattos Neto*

Recife

2019

*Para meu irmão Caio Burgardt.*

## **AGRADECIMENTOS**

Gostaria de agradecer a minha mãe Cristiane por ser meu porto seguro durante as tempestades da minha vida. É nela que encontro sempre a calmaria para meu coração inquieto.

Gostaria de agradecer ao meu pai e conselheiro de vida Otávio por ser um grande amigo e meu exemplo de retidão.

Gostaria de agradecer ao meu irmão e meu melhor amigo Caio que me ensinou tudo sobre segurança da informação e com sua generosidade tem sempre a palavra certa para me tornar uma pessoa melhor.

Gostaria de agradecer a todos os meus professores e em especial ao professor Doutor Paulo Salgado que foi o grande responsável por despertar em mim o interesse por Ciência dos Dados, além de me orientar durante todo processo de construção deste trabalho.

Gostaria de agradecer a minha namorada e psicóloga favorita Ana Carolina pela paciência durante minha ausência nas noites de estudo, além de ser um Anjo em minha vida.

*True CyberSecurity is preparing for what's next, not what was last.*  
—NEIL RERUP

## RESUMO

À medida que os dispositivos móveis baseados em *Android* se tornam cada vez mais populares, cresce o número de aplicativos que lidam com dados financeiros e pessoais. Esse fato mostrou ao mercado do crime cibernético que o ecossistema *Android* é um alvo muito atraente e rentável. Por esse motivo, os fraudadores estão produzindo mais aplicativos maliciosos compatíveis com essa plataforma. Hoje, analisar *Malwares* de forma individual é inviável pelo gasto de tempo com essa tarefa e pela grande quantidade de aplicativos lançados diariamente no mercado digital. Assim, torna-se indispensável fazer o uso de detectores automáticos de “Programas Maliciosos”. Neste trabalho foi feita uma análise de detectores de *Malwares* baseados em aprendizagem de máquina com foco em *Android*. Agrupados 5079 aplicativos maliciosos coletados de três bases de *Malwares*, foram extraídas diferentes *features* dos aplicativos coletados e utilizadas como insumo para criação de diferentes modelos de aprendizagem de máquina para detecção. A partir desses testes foi constatado que o uso de “*intents*” dos aplicativos com *Random Forest* conseguem produzir bons resultados na detecção de *Malware* em *Android*.

**Palavras-chave:** *Malwares*. *Android*. Segurança em Smartfones. Detecção de *Malwares* e Análise de *Malwares*.

## ABSTRACT

As mobile devices built on Android become increasingly more popular, the number of apps that handle financial and personal data increases. This fact showed the cybercrime market that the Android ecosystem is a very attractive and profitable target. For this reason, fraudsters are focusing on producing more malicious applications compatible to this platform. Today analyzing malware individually is not feasible, because the task spends a lot of valuable time besides the large number of applications being daily launched in digital markets. Thus, it is indispensable to make use of automatic detectors of "Malicious Programs". This undergraduate work aims to study the importance of attribute selection for Malware detectors on Android smartphones. By bundling 5079 malicious applications collected from three Public Malware databases, different features were extracted from the collected applications and used as input for creating different machine learning models for detection. From these tests it was found that using "intents" of applications with Random Forest can produce good results in detecting Malware on Android.

**Keywords:** Android. Malware. Smartphone Security. Malware Detection and Malware Analysis.

## LISTA DE ILUSTRAÇÕES

Figura 1.1 - Gráficos de barra representando evolução e crescimento de <i>Malwares</i> em dispositivos móveis de 2016 até 2018.....	12
Figura 2.1 - A pilha de software do <i>Android</i> .....	15
Figura 2.2 - Representação que mostra a ascenção do Android ao domínio dos Smartfones.....	17
Figura 2.3 - Componentes de um arquivo .APK.....	18
Figura 2.4 - Representação da mudança de forma de aprovação de permissão pelo usuário, na esquerda o <i>RunTimeRequest</i> e na direita o <i>InstalTimeRequest</i> .....	20
Figura 2.5 - Pilares da Ciéncia dos Dados.....	22
Figura 2.6 - Representação do ciclo de vida da produção de <i>Malware</i> .....	27
Figura 2.7 - Taxonomia de detectores de <i>Malware Android</i> existentes.....	32
Figura 2.8 - Transformação de <i>Malware</i> em imagens <i>Grayscale</i> .....	34
Figura 2.9 - Um exemplo de como funciona o detector <i>VirusTotal</i> quando detecta um aplicativo malicioso e o relatório que ele retorna.....	36
Tabela 3.1 - Base disponíveis online .....	41
Figura 3.1 - Exemplo de comando utilizando script AZ em que é solicitado 10 aplicativos iniciando na data 11/12/2015 de tamanho de até 3.000.000 bytes que foram publicados nos mercados play.google.com ou appchina. ....	42
Tabela 3.2 - Tabela de Base de Dados Bruta .....	43
Figura 3.2 - Código suporte <i>MakeCSV.py</i> .....	45
Figura 3.3 - Código <i>Extractor.py</i> que extrai os recursos dos aplicativos APK .....	46
Tabela 3.3 - Base de Dados extraídos bruta .....	47
Figura 3.4 - Código ‘ <i>Preprocessamento.ipynb</i> ’, representado em corte do início e fim.....	48
Tabela 3.4 - Bases de dados extraídos após pré processamento e tratamento dos dados.....	49
Figura 3.5 - Código ‘ <i>AI.ipynb</i> ’ , representado em corte início e fim .....	50
Figura 3.6 - Exemplo de matriz de confusão gerada a partir do IA.ipynb .....	51
Tabela 3.5 - Tabelas correspondente aos resultados da execução do AI.ipynb em cada uma das bases pré processadas.....	52
Tabela 3.6 - Tabelas que possuem o índice de falso positivo e falso negativo de cada modelo, as médias de cada classificador e as médias dos resultados. ....	54
Tabela 3.7 - Tabela da Precisão Média de <i>Features</i> .....	56
Tabela 3.8 - Tabela da Precisão Média de Algoritmo.....	56
Tabela 3.9 - Tabela de Média de FN em <i>Features</i> .....	57
Tabela 3.10 - Tabela de Média de FN em <i>Features</i> .....	57
Tabela 3.11 - Tabela de Falso Positivo e Negativo entre Algoritmos .....	57

## LISTA DE ABREVIATURAS E SIGLAS

APK -	<i>Android Package</i>
ART -	<i>Android Runtime</i>
AUROC -	<i>Area Under the Receiver Operating Characteristics</i>
CSV -	<i>Comma Separated Values</i>
FN -	<i>False Negative</i>
FP -	<i>False Positive</i>
GPS -	<i>Global Position System</i>
HAL -	<i>Hardware Abstraction Layer</i>
IA -	Inteligencia Artificial
JAR -	<i>Java Archive</i>
KNN -	<i>K Nearest Neighbor</i>
PC -	<i>Personal Computer</i>
RF -	<i>Random Forest</i>
ROC -	<i>Receiver Operating Characteristics</i>
SMS -	<i>Short Message Service</i>
SO -	Sistema Operacional
SVM -	<i>Support Vector Machines</i>
TI -	Tecnologia Informação
TN -	<i>True Negative</i>
TP -	<i>True Positive</i>
UNB -	<i>University of New Brunswick</i>
URL -	<i>Uniform Resource Locator</i>
VM -	<i>Virtual Machine</i>

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	11
1.1	Motivação .....	12
1.2	Objetivos .....	13
<b>2</b>	<b>CONCEITOS BÁSICOS .....</b>	14
2.1	Android .....	14
2.1.1	<b>Plataforma <i>Android</i> .....</b>	14
2.1.2	<b>Mercado <i>Android</i> .....</b>	16
2.1.3	<b>O Pacote <i>Android / Android Package (APK)</i> .....</b>	17
2.1.4	<b>Arquivo Manifesto .....</b>	18
2.1.5	<b>Sistema de Permissão .....</b>	19
2.1.6	<b>Sistema de Permissão como mecanismo de Segurança .....</b>	20
2.2	Ciência dos Dados e Segurança .....	21
2.2.1	<b>Definição de Ciência dos Dados e seu contexto em Segurança .....</b>	21
2.2.2	<b>Importância de Ciência dos Dados para Segurança .....</b>	22
2.2.3	<b>Aprendizagem de Máquina .....</b>	23
2.2.4	<b>Algoritmos de Classificação .....</b>	24
2.2.4.1	Arvore de Decisão .....	24
2.2.4.2	<i>K-Nearest Neighbors</i> .....	24
2.2.4.3	<i>Support Vector Machines</i> .....	24
2.2.4.4	<i>Naive Bayes</i> .....	25
2.2.4.5	<i>Random Forest</i> .....	25
2.3	<i>Malware</i> .....	25
2.3.1	<b>Definição .....</b>	25
2.3.2	<b>Ciclo de Vida do <i>Malware</i> .....</b>	26
2.3.3	<b>Tipos de <i>Malware</i> .....</b>	28
2.3.4	<b>Família de <i>Malware</i> .....</b>	29
2.3.5	<b>Gerações de <i>Malware</i> .....</b>	29
2.3.6	<b>Análise de <i>Malware</i> .....</b>	31
2.4	Detectores de Malware .....	31
2.4.1	<b>Detecção de <i>Malware</i> .....</b>	32
2.4.2	<b>Tipos de Detecção de Malware .....</b>	32
2.4.2.1	<i>Signature Based Approach</i> .....	33
2.4.2.2	<i>Permission based detection</i> .....	33
2.4.2.3	<i>Dalvik Bytecode Analysis</i> .....	33
2.4.2.4	<i>Anomaly Based Detection</i> .....	33

2.4.2.5	<i>Emulation Based Detection</i> .....	34
2.4.2.6	<i>Taint Analysis</i> .....	34
2.4.2.7	Outras Formas .....	34
<b>2.4.3</b>	<b><i>Online Malware Detectors</i></b> .....	35
<b>2.4.4</b>	<b>Aprendizagem de Máquina para detecção de <i>Malware Android</i>.....</b>	36
2.5	O Estado da Arte para Detecção Automatizada de <i>Malware</i> .....	37
<b>2.5.1</b>	<b>Soluções com Análise Estática .....</b>	37
<b>2.5.2</b>	<b>Soluções com Análise Dinâmica .....</b>	38
<b>3</b>	<b>EXPERIMENTO .....</b>	40
3.1	Bases de Dados .....	40
<b>3.1.1</b>	<b>Classificação das Bases .....</b>	40
<b>3.1.2</b>	<b>Bases de <i>Malware Android</i> Selecionadas .....</b>	41
<b>3.1.3</b>	<b>Base de dados de dados brutos .....</b>	43
3.2	Extração de <i>features</i> .....	43
<b>3.2.1</b>	<b>Metodologia de extração .....</b>	44
<b>3.2.2</b>	<b>Código de Extração .....</b>	45
<b>3.2.3</b>	<b>Resultado da Extração .....</b>	47
3.3	Pré processamento de Dados .....	46
3.4	Técnicas de Aprendizagem de Máquinas .....	49
<b>3.4.1</b>	<b>Resultados .....</b>	51
<b>3.4.2</b>	<b>Análise dos Resultados .....</b>	54
<b>4</b>	<b>CONCLUSÃO.....</b>	57
4.1	Trabalhos Futuros .....	57
	<b>REFERÊNCIAS .....</b>	58
	<b>ANEXOS .....</b>	63

## 1. INTRODUÇÃO

Na atualidade os smartphones se tornaram um dos equipamentos mais importante em nosso cotidiano. Entre os fatores que impulsionaram esse cenário além do avanço da tecnologia, estão, o crescimento explosivo na produção de smartphones e o surgimento de novas empresas com estratégias agressiva de venda, que aumentaram a variedades de produtos no mercado com baixo custo [CF]. Outro fator que incentivou esse crescimento é a quantidade de aplicativos que surgem diariamente graças a facilidade de publicar *Apps* em lojas online. Só em 2016 foram baixados 82 bilhões de aplicativos na *Playstore* [STAb].

Outra área que também teve um rápido crescimento por causa dos avanços na tecnologia da informação foi o cibercrime, impulsionado pelo crescimento e disseminação da internet e pelo surgimento e propagação dos smartphones. O fato de cada pessoa ter um dispositivo particular que carrega dados sensíveis criou novas oportunidades de fraude e aumentou a superfície de ataque. Além do fato de que hoje a luta contra cibercrime apresenta novos desafios para os países, pois deve-se adaptar leis para lidar com esses crimes que não acontecem no meio físico, mas em um ambiente virtual que engloba o mundo e oferece aos fraudadores novas possibilidades de anonimato e ocultação [YS19]

*Malwares* são uma ameaça para a informática, pois existe uma lacuna na sua compressão pela falta de entendimento sobre o seu funcionamento, propagação, prevenção e detecção. Por causa desse cenário houve um aumento na procura e nas vendas de software maliciosos dentro do mercado do crime cibernético, o que ocasionou num aumento de pessoas programando softwares maliciosos para venda. Hoje, o que mais se busca dentro dos grupos de venda de *Malwares* são aqueles voltados para smartphones, por causa do número de usuários e pela quantidade de aplicativos com dados sensíveis existentes [AVW<sup>+11</sup>]

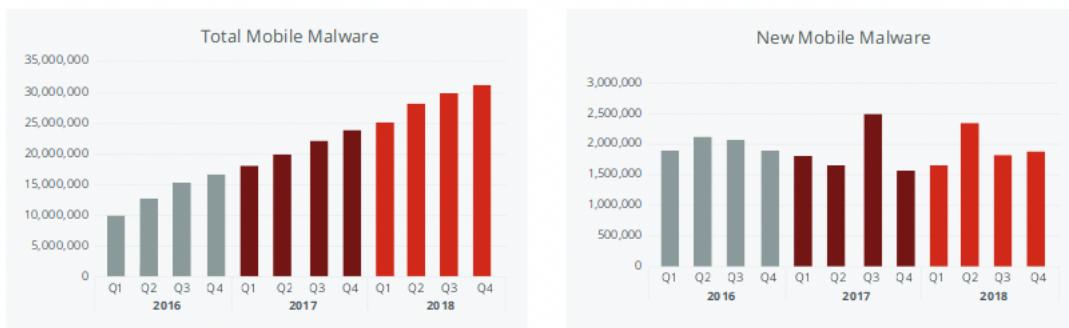
Neste momento, a principal defesa para os smartphones das vítimas são os antivírus produzidos por empresas especializadas em segurança como a Norton, Avast, McAfee, dentre outras. A maioria desses produtos utilizam métodos clássicos de detecção como os baseados em assinaturas. Porém, esses métodos se mostram pouco efetivos para a indústria de *Malwares* que sempre está inovando e encontrando novas maneiras de burlar os sistemas automatizados de detecção. Para melhorar o combate contra os softwares maliciosos, novos detectores baseados em heurísticas foram propostos utilizando grande bases de dados para criar modelos com objetivo de identificar padrões em arquivos maliciosos [HYSA17].

Esse trabalho de graduação busca avaliar características diferentes de softwares malicioso de *Android* que é o sistema operacional que possui maior *Market Share* dos smartphones.

## 1.1 MOTIVAÇÃO

Atualmente encontramo-nos em um momento de grande crescimento da plataforma *Android*, em que essa superou seus concorrentes dominando o mercado de Sistemas Operacionais (SO) de dispositivos moveis, tornando-se o mais popular dos SO com 86,8% de participação no mercado em 2018. Ou seja, para cada 10 donos de smartphones 8 possuem dispositivos *Android* [IDC]. De qualquer forma, hoje, com a usabilidade, *Apps* com funções atrativas, disseminação da tecnologia, barateamento de smartphones e disponibilidade da internet em qualquer lugar, os dispositivos móveis e seus aplicativos se tornaram um grande facilitador das tarefas do dia a dia, passando a se tornar uma necessidade para a vida moderna. Tanto que o número de pessoas no mundo que usam smartphones aumentou em 100 milhões a mais em 2018, atingindo um total de 5.1 bilhões de usuários em janeiro de 2019, fazendo com que 67% da população mundial possua um dispositivo “smart” [Dig].

Porém smartphones não estão mais tão seguros como pensávamos. O rápido crescimento e participação do *Android* no mercado tornou os usuários desses dispositivos alvos atraentes para cibercriminosos gerando um crescimento explosivo na criação e sofisticação de *Malwares Android*. Hoje pode-se observar um ritmo alarmante na produção de *Malwares* através dos gráficos da Figura 1.1.



**Figura 1.1:** Gráficos de barra representando evolução e crescimento de *Malware* em dispositivos moveis desde o primeiro quadrimestre de 2016 até ultimo quadrimestre 2018. Fonte: [RS19]

Com esse cenário em mente, empresas de segurança e pesquisadores acadêmicos estão investindo e esforçando-se ininterruptamente para mudar o ecossistema atual de segurança de

dispositivos móveis, tentando torna-los mais seguro para usuários. Entre os progressos alcançados em maneiras de combater essas ameaças foi criado um dos maiores aliados para combater e analisar aplicativos maliciosos, os detectores automáticos de *Malwares* comumente chamados de Antivírus. Porém, ainda hoje existe muito que precisa ser feito sobre detectores de *Malwares para Android* [QNL<sup>+19</sup>].

## 1.2 OBJETIVO

O objetivo deste trabalho é analisar a importância e influência da escolha de atributos para detectores de *Malware* de smartphones com sistemas *Android*. Serão feitos modelos de detecção utilizando aprendizagem de máquina baseado em informações adquiridas através de análise estática de arquivos “APK”. Os arquivos utilizados serão extraídos de bases públicas de *Malwares*, pois um grande problema encontrado em artigos de detecção de *Malware* em *Android* é a replicabilidade dos experimentos, ou seja, quase sempre são utilizados bases de dados restritas ou privadas que necessitam de recursos financeiros para sua obtenção. Assim, foi adotado a utilização de dados públicos para facilitar a reprodução dos experimentos criados nesta monografia.

## 2. CONCEITOS BÁSICOS

Num primeiro momento, antes de nos aprofundarmos no estudo de detectores de software malicioso em *Android*, temos primeiro que entender uma série de conceitos básicos fundamentais que estão relacionados e compõe a construção de um sistema para detecção automatizada de *Malware* em *Android*. Assim, apresenta-se um capítulo de conceitos básicos de *Android*, ciência dos dados aplicado a segurança, *Malwares* e detectores automáticos de *Malwares*.

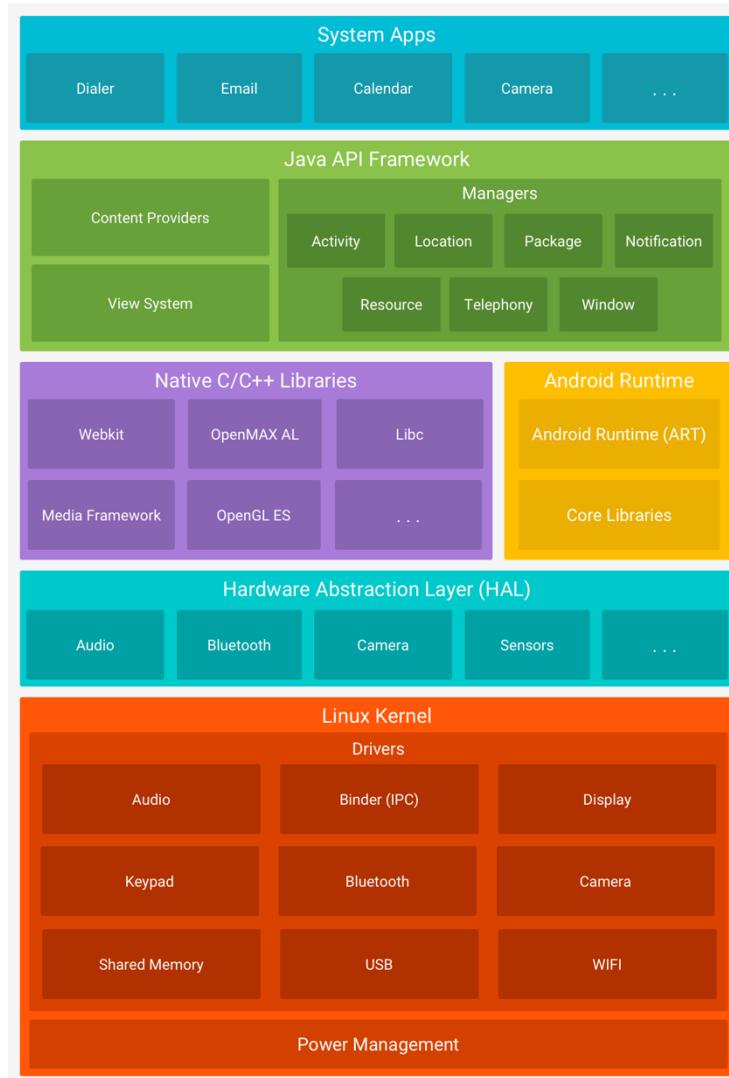
O entendimento correto desses conceitos é fundamental e necessário, pois para realizarmos uma correta análise do problema precisamos partir de um alicerce formatado e sólido do aprendizado, considerando que hoje em dia tem-se uma grande carência no conhecimento de *Malwares* e de como deve-se analisá-los. Dessa forma, as pessoas não sabem da complexidade desses softwares, de como uma pequena mudança no código evita a detecção automática. Ainda, a falta de conhecimento sobre sua classificação, de como é feita a análise de seus artefatos maliciosos e de sua constante evolução, torna o problema de detecção de *Malwares* em um estudo não trivial e complexo [YY10].

### 2.1 ANDROID

Nesta seção, detalha-se conceitos importantes sobre a plataforma *Android*. Entre eles vai-se explicar o sistema operacional, seu crescimento e os componentes relevantes de um “APK” para detecção de *Malware*.

#### 2.1.1 PLATAFORMA ANDROID

*Android* é um sistema operacional baseado em Linux de código aberto projetado para diversos dispositivos que consiste em seis camadas: Kernel do Linux, Camada de abstração de hardware (*HAL*), *Android Runtime (ART) / Dalvik Virtual Machine (VM)*, Bibliotecas nativas, Estrutura da Java API e Aplicativos do sistema. Pode-se ver mais detalhadamente suas funcionalidades na Figura 2.1.



**Figura 2.1:** A pilha de software do *Android*. Fonte: [Anda]

Para melhor entendimento, seguem abaixo as definições de cada camada:

- **Kernel do Linux / Linux Kernel** - essa camada é a base do sistema operacional. Nela temos funcionalidades como gerenciamento de memória, processo e o sistema de segurança baseado em permissão.
- **Camada de Abstração de Hardware / Hardware Abstraction Layer (HAL)** - por causa da falta de padronização de componentes de hardware para interface *Linux* foi criado uma camada HAL que cria uma camada de abstração entre a aplicação e o hardware.

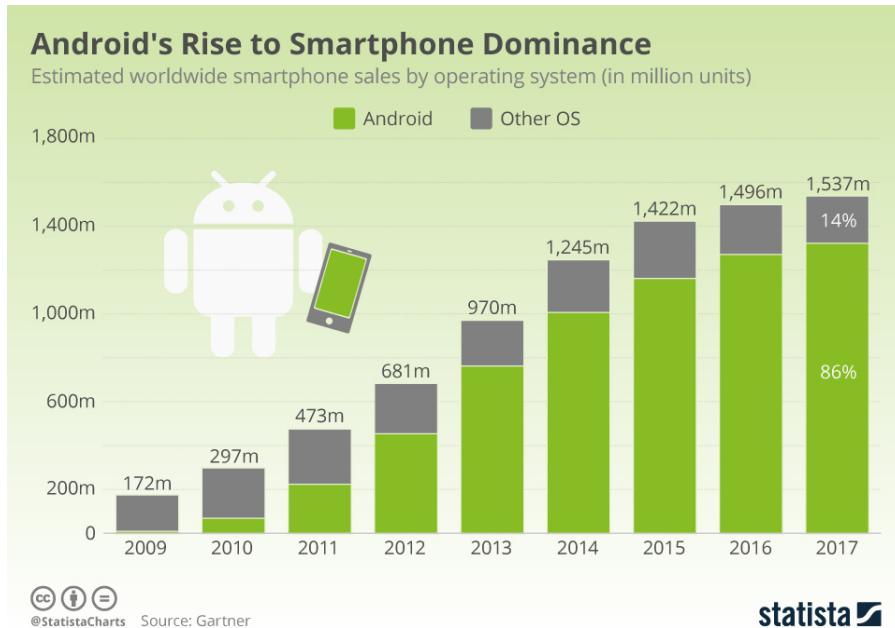
- **Bibliotecas Nativas / Native Libraries** - é a camada que possui bibliotecas programadas em C e C++ que são utilizadas para dar suporte aos componentes e serviços do sistema *Android*.
- **Android Runtime (ART) e Dalvik Virtual Machine (VM)** - é a camada responsável pela representação do ambiente de tempo de execução do aplicativo. Onde cada aplicativo tem sua própria cópia de máquina virtual com ID diferente encapsulando e protegido de outra aplicação. O ART executa várias máquinas virtuais em dispositivos de baixa memória executando arquivos DEX que é um formato de bytecode projetado especialmente para *Android*.

Cabe aqui esclarecer que, em versões mais antigas do *Android* usa-se a *Dalvik Virtual Machine (VM)* ao invés da *Android Runtime (ART)*, que se diferencia dessa principalmente pela técnica de compilação em que o ART executa a compilação antes da execução e o VM utiliza a compilação durante a execução.

- **Estrutura da Java API / Java API Framework** - representa o conjunto de recursos disponíveis no SO *Android* de API feitas em Java. Nele ficam contidas os principais serviços da plataforma que serão utilizados nas aplicações.
- **Aplicativos do Sistema / System Apps** - são os aplicativos que vem junto com o SO *Android* como e-mails, SMS, navegadores pré-instalados, etc.

### 2.1.2 MERCADO ANDROID

O *Android* é um sistema operacional para dispositivos móveis criado a partir de uma colaboração entre a *Google* e *Open Handset Alliance* que corresponde a um aglomerado de organizações comprometidas em melhorar e “abrir” o mercado de dispositivos móveis”. Lançado em 2008 como um software de código aberto, em pouco tempo mudou drasticamente a indústria de smartphones sendo absorvido pelos maiores manufatureiras de celulares no mundo [ASKO11]. Pode-se observar seu crescimento em comparação com seus concorrentes na Figura 2.2.

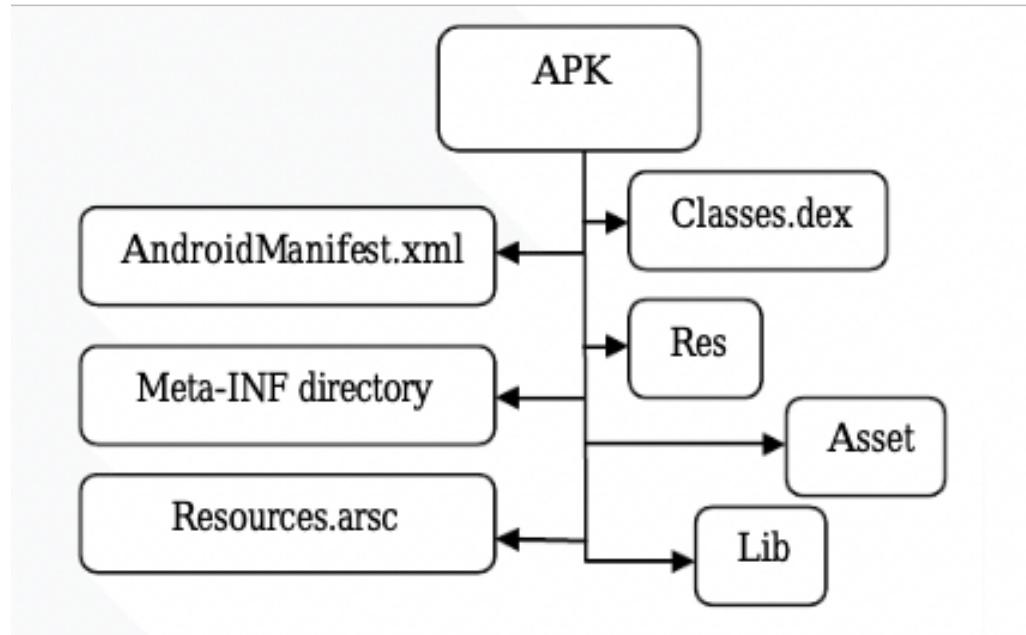


**Figura 2.2:** Representação que mostra a ascensão do *Android* ao domínio dos smartphones. Fonte: [Staa]

Sua evolução e dominação do mercado se dá por vários fatores entre eles os principais são: sua criação ocorreu em um momento oportuno em que os usuários de celulares estavam migrando para smartphones, demandando assim, a criação de melhores serviços e funcionalidades mais complexas; e o fato do sistema ser *Open Source*, em que temos diversas pessoas e companhias qualificadas de qualquer parte do mundo trabalhando para criar funcionalidades e serviços para usuários [ASKO11]

### 2.1.3 O PACOTE ANDROID / ANDROID PACKAGE (APK)

As aplicações para *Android*, programas, são instalados através de arquivos de pacote chamados *Android Application Package* (APK) conforme se vê na Figura 2.3. Esses arquivos possuem a extensão .APK e são semelhantes aos arquivos .EXE (executável) do Windows. De forma resumida, um arquivo APK é um arquivo ZIP (compactado) baseado no formato JAR (programável) que contém arquivos referente ao código da aplicação, dos recursos que serão utilizados e do arquivo manifesto [RFC13].



**Figura 2.3:** Componentes de um arquivo APK. Fonte: [BPK15]

A estrutura de um arquivo APK é a seguinte [BPK15]:

- **Android manifesto** – que corresponde ao arquivo no formato XML que declara o nome do pacote do aplicativo, os componentes da versão, permissões e outros metadados.
- **Classes.dex** - que é o arquivo executável da máquina virtual *Dalvik*.
- **Res** - que é a pasta que contém recursos não compilados em *resources.arsc* como idiomas, configurações de som, layout gráfico, atributos, *drawables*, etc.
- **Meta-INF** - que é uma pasta que contém a assinatura do aplicativo, o seu certificado, além de conter o arquivo MANIFEST.MF, que armazena metadados sobre o conteúdo do JAR.
- **Asset** – que é uma pasta opcional no arquivo .APK que contém ativos que podem ser lido pelo aplicativo usando o *assetManager*.
- **Resources.arsc** – que é um arquivo contendo recursos pré-compilados da aplicação no formato XML.
- **Lib** – que é uma pasta opcional no arquivo .APK que contém bibliotecas nativas.

#### 2.1.4 ARQUIVO MANIFESTO

Como vimos acima, o arquivo manifesto é um dos componentes do *Android Package*, APK, que consiste em um arquivo XML que contém informações sobre o aplicativo como nome, componentes, permissões, recursos utilizados e outros metadados. Dessa forma, o

arquivo manifesto pode ser visto como uma fonte de dados resumida do comportamento do aplicativo servindo como uma boa fonte para uma análise de comportamento do programa sendo utilizado por diversos detectores de *Malwares* [RFC13].

O arquivo manifesto é composto de vários elementos cada um com informações essenciais e importantes sobre o aplicativo dentre eles temos: *<permission>* que serve para configurar as permissões do aplicativo, *<intent-filter>* que define as intenções de uma atividade, serviço ou um *broadcast receiver* (*receptor de mensagens*), *<provider>* que declara um *ContentProdiver* (*provedores de conteúdo*), *<receivers>* especifica um componente do *broadcast receiver*, e *<services>* que especifica serviços como um dos componentes do aplicativo entre outros recursos[LLH<sup>+16</sup>].

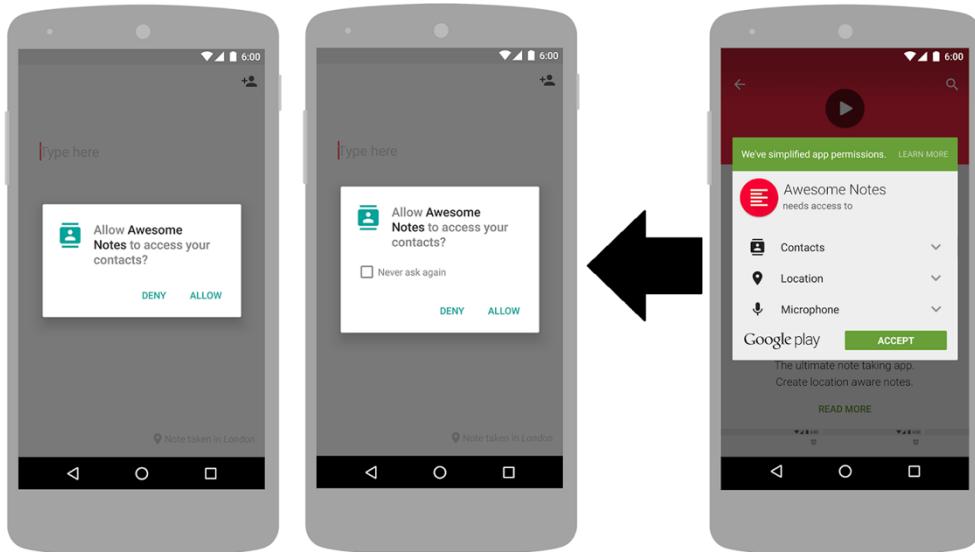
## 2.1.5 SISTEMA DE PERMISSÃO

Todos os aplicativos *Android* são executados dentro de um "container" com acesso limitado. Quando são requisitados recursos do usuário como contatos ou quando é necessário acesso a recursos do sistema como Internet, câmera, microfone ou memória, é necessário que o aplicativo defina uma permissão para conseguir acesso a esses recursos. As permissões de um aplicativo são declaradas no seu arquivo manifesto, caso contrário, a aplicação não irá ter acesso aos recursos desejados. O objetivo do uso de permissões no sistema *Android* é feita para proteger a integridade das outras aplicações do próprio sistema e da privacidade do usuário [andb].

Segundo a documentação de desenvolvimento oficial do *Android* temos dois tipos de permissão: Permissão Normal - que corresponde a permissões que não apresentam risco ao usuário ou ao sistema; e a Permissão Perigosa - que corresponde as permissões que tem potencial para comprometer o sistema ou privacidade do usuário [andb].

A aprovação das permissões no *Android* passou por mudanças com a evolução do sistema operacional conforme pode-se observar na Figura 2.4. Diferentemente das permissões normais que são aprovadas automaticamente pelo SO as permissões perigosas devem passar pela aprovação do usuário, que pode ser antes de instalar a aplicação ou durante o seu tempo de execução. A aprovação das permissões nas versões mais antigas (abaixo do SO *Android* 5.1.1) ocorre através do '*Install-time requests*' em que o sistema irá pedir aprovação do usuário para ter acesso a todas as permissões perigosas antes de instalar o aplicativo, que caso negado, o

aplicativo não será instalado. Nas versões mais novas tem-se um novo método de aprovação o ‘*Runtime requests*’ em que o aplicativo pede aprovação de uma permissão perigosa apenas quando for utilizar uma funcionalidade ligada a esse. Caso o usuário negue a permissão, ele não executará uma das funcionalidades da aplicação [WGNF12].



**Figura 2.4:** Representação da mudança na forma de aprovação de permissão pelo usuário em que na esquerda tem-se as versões mais novas, *Runtime requests* e na direita tem-se *Install-time requests*. Fonte: [andb]

## 2.1.6 SISTEMA DE PERMISSÃO COMO MECANISMO DE SEGURANÇA.

A plataforma *Android* fornece várias medidas de segurança para proteger o usuário de instalar aplicativos maliciosos, a mais notável dessas medidas é o modelo de segurança baseado em permissões que media acesso a recursos sensíveis.

Esse mecanismo de segurança no *Android* é muito bem pensado na teoria, pois os softwares maliciosos tendem a usar algumas permissões perigosas, mas que outras, como é o caso da permissão para acesso a envio de SMS “*Send SMS Permission*” em que a grande porcentagem de *Malwares* utilizam SMS para enviar dados para os cibercriminosos [ASH<sup>+14</sup>].

Porém, na prática, o sistema é falho, pois grande parte dos usuários aprovam cegamente permissões para aplicativos não confiáveis, destruindo o propósito de permissões como um mecanismo de segurança. Dessa forma, o sistema de permissões não é efetivo no combate do funcionamento de *Malwares* no sistema *Android* [Tch14].

Além disso, esse mecanismo chamou a atenção cibercriminosos que criaram novas formas de atacar as vulnerabilidades do sistema como: ataques de escalar privilégios, ataques de redelegação entre outros [WGNF12].

## 2.2 CIÊNCIA DOS DADOS E SEGURANÇA

Nesta seção encontram-se alguns conceitos chave para entender os algoritmos que envolvem os experimentos desta monografia e definições para facilitar o seu entendimento sobre os sistemas de detecção de *Malwares*. Assim, define-se o conceito de Ciência dos Dados e logo após o seu uso em segurança da informação, aprendizagem de máquina e um noção básica dos algoritmos que serão utilizados nos experimentos deste trabalho.

### 2.2.1 DEFINIÇÃO DE CIÊNCIA DOS DADOS E SEU CONTEXTO EM SEGURANÇA

A Ciência dos Dados é um conceito relativamente novo, porém com raízes em trabalhos antigos como os cálculos de expectativa de vida da população realizados por *John Graunt* em 1662. Na verdade, encontram-se várias definições em que de fato significa esse campo de estudo, mas após ler diversas definições extrai o núcleo principal e fiz uma definição resumida do assunto [VDA16].

Do entendimento, define-se, resumidamente, que Ciência dos Dados é uma área multidisciplinar que utiliza algoritmos e métodos estáticos em um conjunto de dados para extrair *insights* dos dados com objetivo de solucionar problemas relevantes.

Esse campo de estudo pode ser subdividido em 3 pilares conforme verifica-se na Figura 2.5: Aprendizagem de Máquina, Mineração de Dados e Visualização de Dados. Tendo em mente o contexto de detecção de *Malwares*, Aprendizagem de Máquina se refere aos algoritmos superiores às técnicas tradicionais que irão aprender com a base de dados e serão capazes de detectar ameaças. Mineração de Dados, por sua vez, em detecção de *Malware*, refere-se aos métodos capazes de achar padrões de ameaças dentro da base de dados e, Visualização de Dados, refere-se a formas inovadoras de visualizar ameaças e grupos com características semelhantes (Famílias) [SS18].



**Figura 2.5:** Pilares de ciência dos dados. Fonte: [SS18]

### 2.2.2 IMPORTÂNCIA DE CIÊNCIA DOS DADOS PARA SEGURANÇA

Por causa do grande crescimento da Tecnologia da Informação (TI) foi proporcionado a criação de produtos complexos, que por definição são aqueles que geram grande quantidade de dados, em abundância e são utilizados por muitas pessoas e lidam com dados sensíveis, como redes sociais, sistemas bancários, sistemas educacionais e do governo [SS18]. Todo esse crescimento também gerou um grande desenvolvimento na área de cibersegurança, que tem o objetivo manter esses sistemas complexos seguros, protegidos de fraude e com privacidade.

Assim, a segurança da informação evoluiu bastante nos últimos anos e nesse último período, se caracterizou pela multidisciplinaridade da área em que cada vez mais é aplicado técnicas de aprendizado de máquina e de mineração de dados para aprimorar, projetar e desenvolver novos sistemas de segurança [DD16].

Mineração de Dados e Aprendizagem de Máquina passaram a ter um papel crucial para cibersegurança, pois atualmente os métodos e abordagens tradicionais utilizados são praticamente inviáveis, considerando-se a rápida transformação do ambiente. Assim, fatores como a necessidade de descoberta rápida da informação, a quantidade e diversidade de dados gerados, a mudança frequente de ameaças e seus ataques, geraram a necessidade de automatização de tarefas, ou seja, os sistemas precisam realizar autonomamente tarefas para manterem o seu nível de proteção [DD16].

Hoje, a Ciência dos Dados está sendo aplicada em diversas áreas da segurança como: sistemas de intrusão a detecção; detecção de anomalias em sistemas; antivírus de sistemas operacionais; visualização de ameaças; modelos de preservação de privacidade; dentre outros.

### 2.2.3 APRENDIZAGEM DE MÁQUINA

Aprendizagem de máquina é um campo da inteligência artificial que surgiu graças a junção, evolução, de duas áreas de pesquisa, o reconhecimento de padrões e a teoria do aprendizado computacional. O foco da aprendizagem de máquina é fornecer aos sistemas a capacidade de aprender automaticamente e independentemente a partir de algoritmos e da observação de dados, ou seja, é treinar o sistema a reconhecer os padrões e o comportamento dos dados [Har12].

O processo de Aprendizagem de Máquina (*Machine Learning*) começa através do *input* de uma base de dados limpa em que o sistema observa essa base e utiliza um algoritmo de aprendizagem, preparando um modelo para a solução do problema. Existem quatro técnicas possíveis em aprendizagem de máquina: aprendizagem supervisionada; não-supervisionada, semi-supervisionada e reforçada. Assim, tem-se:

- Aprendizagem supervisionada o *input* da base de treinamento vem com os dados classificados (*labeled data*);
- Não supervisionada não existe *labels* o programa descobre e explora sozinho a estrutura dos dados;
- Semi-supervisionada é a combinação das duas anteriores existindo dados classificados e outros não na base de treinamento; e
- Aprendizagem Re却orçada o método é feito através da interação do sistema com ambiente dessa forma o programa aprende sozinho como se comportar de forma ideal no ambiente.

Para executar algoritmos de aprendizagem de máquina, primeiro deve-se fazer uma série de tratamentos na base a ser utilizada, dentre esses tratamentos, um dos mais importantes, é a divisão da base em treino e teste. Essa divisão pode ser feita de forma arbitrária, porém corre-se o risco de se ter sorte e azar nessa divisão ou pode-se utilizar ‘*cross validation*’ que é uma técnica que ajuda na generalização de um modelo. Uma das formas para fazer a *cross validation*

é através de *K-fold* que é uma técnica em que a base de dados é dividida em um total “k” subconjuntos em que cada subconjunto será teste e treino em um dado momento [RTL09].

## 2.2.4 ALGORITMOS DE CLASSIFICAÇÃO

Nesta secção será esclarecido, brevemente, os algoritmos de aprendizagem de máquina que são utilizados no código de aprendizagem do experimento e outros detalhes importantes.

### 2.2.4.1 ÁRVORE DE DECISÃO

Árvore de Decisão é um dos algoritmos de classificação mais utilizados em uma estrutura de dados, funcionando da seguinte maneira, para criar uma árvore de decisão primeiro deve-se calcular todas as entropias das *features* (características) que é a capacidade de dividir os dados da base, após esse cálculo é escolhido o melhor resultado encontrado. Esse processo de cálculo para divisão acontece em toda árvore até que sejam classificados todos os dados, ou seja, na base de treinamento deve-se executar esse processo da raiz até as folhas, sendo essas o conjunto de dados representado apenas por uma classe [Har12].

### 2.2.4.2 *K-NEAREST NEIGHBORS*

*K-Nearest Neighbors* é um algoritmo que pode ser usado para classificação de base de dados usando o conceito do cálculo de distâncias entre itens que funciona da seguinte forma: existe um conjunto de treinamento em que se conhece as classes aos quais esses dados pertencem, sendo definidos em um espaço considerando seus atributos. Dessa forma, quando se recebe um dado novo, sem classe, compara-se esse dado com todos os dados existentes, calculando-se as distâncias euclidianas entre todos os itens. Em seguida é classificado o dado desconhecido baseado nos seus K vizinhos mais próximos, que é um numero definido pelo usuário. Esses K vizinhos votam baseados na própria classe ao qual deve ser a classe do novo item e o voto da classe majoritária define a classe do novo item [Bur19].

### 2.2.4.3 *SUPPORT VECTOR MACHINES (SVM)*

*Support Vector Machines (SVM)* podem ser utilizados para classificação de dados funcionando da seguinte forma: são plotados os dados da base treino em um espaço n-dimensional em que “n” é o numero de *features* (características). Depois é calculado hiperplanos que sejam capazes de separar as classes da melhor forma possível, logo após é

escolhido o melhor hiperplano e quando queremos descobrir a classe de um conjunto específico de dados, plota-se esses dados nesse espaço com hiperplanos, ficando fácil a sua definição [Har12].

#### 2.2.4.4 NAIVE BAYES

*Naive Bayes* é uma forma de classificação que se baseia em cálculos matemáticos de probabilidade usando a teoria da decisão bayesiana. De forma resumida, o algoritmo funciona da seguinte maneira: é calculado a probabilidade de cada classe dado as suas features (características) e depois é comparado os resultados e escolhido aquele com maior probabilidade de um conjunto de dados pertencer a uma classe [Har12].

#### 2.2.4.5 RANDOM FOREST (RF)

*Random Forest (RF)*: é uma forma de classificação baseada no uso de árvores de decisão e sistemas de votação. O algoritmo funciona da seguinte forma: é criado um conjunto de árvores de decisão utilizando partes da base de treinamento que depois de pronto, quando queremos classificar um novo dado, esse é classificado por todas as árvores que votam para ver qual devia ser a classe do novo dado, o voto majoritário define a classe do novo dado[Bur19].

### 2.3 MALWARES

Nesta seção se introduz as noções básicas sobre softwares maliciosos dentre elas a definição de *Malware*, seu comportamento, o ciclo de vida ou produção de um *Malware*, como classificar um *Malware* de acordo com seu tipo, família e geração. Além disso, também se encontra como é feita a análise desses aplicativos maliciosos e as peculiaridades que envolvem esta análise.

#### 2.3.1 DEFINIÇÃO DE MALWARES

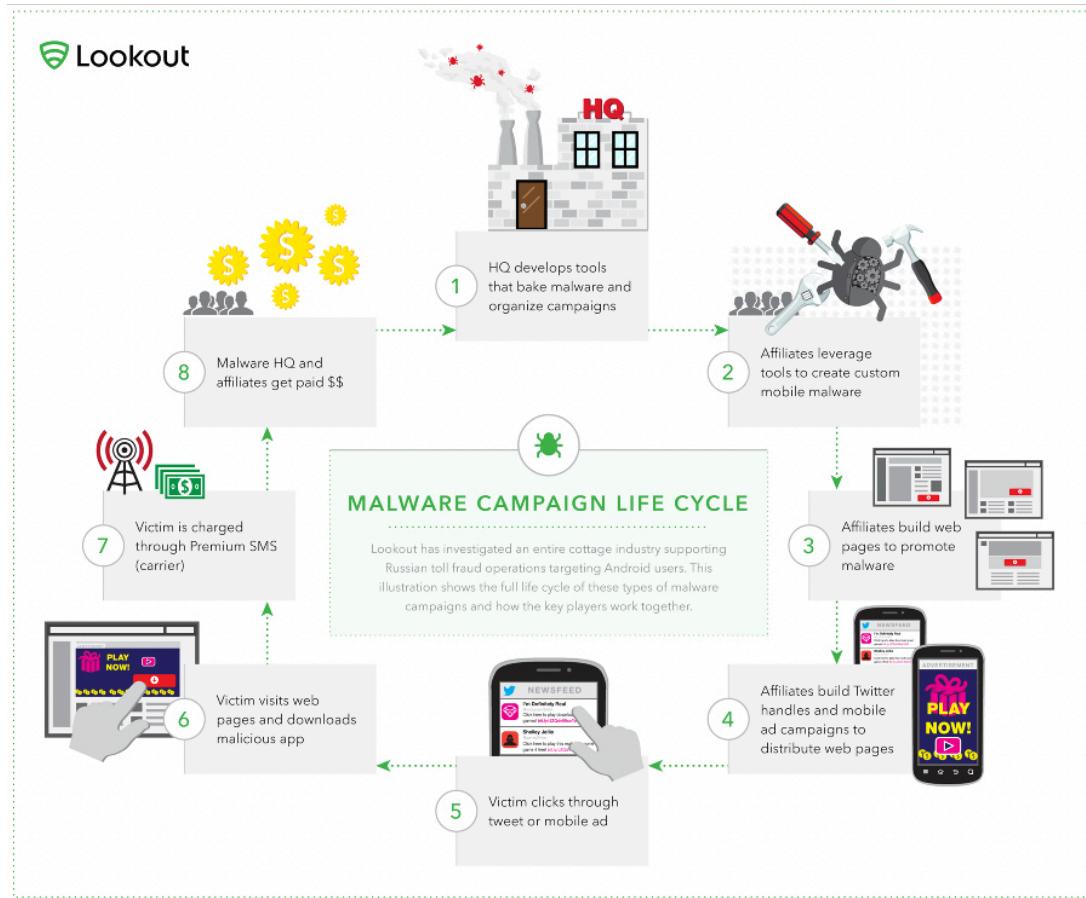
*Malware* ou software malicioso é definido como qualquer software que foi criado com intutos maliciosos e tem como principal objetivo prejudicar a vítima causando danos ao computador, servidores, clientes, privacidade ou a rede. *Malwares* geralmente são criados com fins lucrativos por grupos de *hackers* que após produzi-los, vendem esses programas em grupos de fraudadores, criminosos ou estelionatários, sendo as vezes disseminados por eles mesmo no ambiente cibernético. Quando o *Malware* é produzido sem objetivo de lucrar, geralmente são

desenvolvidos por agentes governamentais, grupos terroristas, ou protestantes, que utilizam o artefato como uma arma debilitadora ou como uma forma de protesto [LKS17].

É importante relembrar que *Malwares* não são exclusivos apenas a computadores, mas também podem ser encontrados em dispositivos móveis em que o software malicioso tenha acesso aos componentes disponíveis, como câmera, GPS, microfone, lista de contatos, entre outros. Hoje em dia são encontrados mais facilmente *Malwares* em dispositivos móveis *Android* que em outras plataformas, esse fato está relacionado ao crescimento do sistema operacional *Android* e ao número de aplicativos produzidos diariamente.

### 2.3.2 CICLO DE VIDA DO MALWARES

Para entender como funciona e o que um *Malware* é capaz de fazer devemos primeiro entender o seu ciclo de vida. Na figura 2.6. tem-se o seu ciclo, sendo o seu início marcado pelo seu desenvolvimento executado pelos seus autores criminosos, em que são utilizados os conhecimentos tecnológicos para a criação do software malicioso. A segunda etapa é a sua disseminação pelos dispositivos e pelas redes, que pode ser feita de forma física, através de portas USB como exemplo, ou virtual, com o envio de e-mail, *phishing*, engenharia social, ou até ataque direto por *hackers*. Após a infecção a vítima está a mercê das funcionalidades do *Malware* que pode roubar o número de seu cartão de credito, da senha de bancos, senha de lojas e comodidades de serviços, e enviar um SMS para um agente malicioso com dados privados, possibilitando a chantagem ou o roubo programado. Assim, o grupo de *hackers* mantém o ciclo rentável de desenvolvimento recebendo recurso e reaplicando em novos desenvolvimentos [fra].



**Figura 2.6:** Representação completa do ciclo de vida da produção de *Malwares*. Fonte: [fra]

Em se tratando de *Malwares* criados para dispositivos móveis o ciclo de vida se diferencia apenas na parte da dispersão/disseminação do aplicativo malicioso e em sua penetração, em que esse pode ser disseminado em lojas falsas ou não confiáveis, ou através de *phishings*, golpes de engenharia social ou até em lojas oficiais. Quanto às táticas de penetração do sistema podem ser por *Repackaging* em que cibercriminoso pega um software legítimo, faz a sua desmontagem (*disassemble*) e adiciona o *Malware* em sua estrutura e refaz o *Repackaging* do software alterado e sua nova distribuição, ou por *Updating*, em que o cibercriminoso infecta um código benigno e adiciona um componente de *update* para fazer um download do *Malware* junto com o sistema benigno, sendo a vítima enganada quando realizar a atualização no sistema [SS14].

### 2.3.3 TIPOS DE *MALWARES*

Para facilitar e organizar os estudos na área de *Malware* pesquisadores classificam os artefatos maliciosos encontrados em tipos de *Malwares*. A classificação pode ser feita de acordo com uma série de características como: o comportamento, a características do código, a sua distribuição, a plataforma ou dispositivo que foi destinado, o objetivo do *Malware*, a forma de contágio, as vulnerabilidades que ele utiliza e a metodologia de ataque. Após o pesquisador descobrir as características do *Malware* analisado, ele classifica o software malicioso dentre os tipos existentes. É importante lembrar que um *Malware* pode se comportar como vários tipos diferentes, então o pesquisador escolherá qual tipo melhor se encaixa para aquele específico programa [GAF<sup>+</sup>15].

- **Vírus** - A população está acostumada a chamar todos os *Malwares* de Vírus, porém esse termo apenas se designa a um tipo específico de *Malware*. O vírus é um software que infecta arquivos legítimos do sistema e os modifica de uma forma que se ao executar o arquivo legítimo, você executara o vírus também, conjuntamente. Os Vírus eram mais presentes antigamente, porém hoje correspondem a 10% dos *Malwares* espalhados [typ].
- **Worms** - Correspondem a um tipo de *Malware* que tem o poder de se replicar e infectar outras máquinas utilizando a rede da vítima. *John von Neumann* teorizou a ideia de programas que poderiam se replicar, porém só nos anos 70 exemplos desse tipo de *Malware* começaram a aparecer [typ].
- **Trojan** - É um programa malicioso que se passa por um legítimo, porém contém instruções maliciosas. O nome “trojan” é baseado no mito grego do cavalo de tróia que se assemelha muito com os *Malwares* em que a escultura do cavalo representa o arquivo aparentemente inofensivo e dentro dele, os soldados que são as instruções maliciosas. Os cibercriminosos iludem as vitimas através de e-mails, engenharia social e *phishing* para instalarem o arquivo *trojan* [mlr].
- **Ransomware** - É um tipo de *Malware* que ficou muito popular recentemente em que o cibercriminoso criptografa o computador da vítima e faz seus dados de refém em busca de dinheiro de resgate. Um *ransomware* funciona da seguinte forma, após a infecção o software malicioso trava o computador da vítima e pede uma chave para retornar o estado anterior em troca de um valor de resgate na conta bancaria do fraudador. Vale

ressaltar que na atualidade a maioria das transferências de dinheiro para os autores de *Malwares Ransomware* é feito através de *bitcoins* [mlr].

- ***Adware*** - Diz respeito a um tipo de *Malware* que foi projetado para exibir anúncios no seu computador, redirecionar pesquisas de um *browser* (*safari*, *chrome*, *opera*, *ubumto*, *etc*) e coletar dados da vítima. A monetização de *adware* geralmente é feita através de duas formas, os cibercriminosos ganham um valor sobre a quantidade de vezes que os *adware* são expostos ou pela quantidade de “clicks” que seus redirecionamentos geraram [typ].
- ***Rootkit*** - É um dos tipos de *Malware* mais nocivos que existem, pois interagem diretamente com os sistemas operacionais. Esses programas correspondem a um arquivo malicioso que consegue interagir com o sistema operacional, como por exemplo: dar privilégios a arquivos sem autorização, esconder arquivos e alterar funções do SO. Quando uma máquina é infectada por um *Rootkit* é aconselhado formatar o sistema, pois é muito difícil determinar até que parte do sistema operacional foi afetada e infectada por esse tipo de *Malware* [MRB19].

#### 2.3.4 FAMÍLIA DE MALWARES

O desenvolvimento do *Malware* passa pelo mesmo processo de construção que um software benigno como fase de protótipos, testes, resistência à antivírus, adicionar funções, etc. Sendo assim, tem-se várias versões do mesmo artefato malicioso como alpha, beta ou “v1”. No estudo de *Malware* os pesquisadores se referenciam ao grupo de versões do ciclo de desenvolvimento do software malicioso como “Família”. Outro caso em que um *Malware* é associado a uma família é se ele utiliza a mesma engenharia de mitigação de vulnerabilidade e comportamento semelhante [SWL12].

#### 2.3.5 GERAÇÕES DE MALWARES

Desde a criação dos primeiros *Malwares* no fim dos anos 80, a área da Tecnologia da Informação (TI) mudou muito em comparação com a atualidade, grandes avanços surgiram nas áreas de comunicação, computação em nuvem, internet das coisas, bancos de dados, dentre outras. A área de segurança também teve diversos avanços em detectores e análise de software, porém a medida que avançamos em técnicas de novas detecções, os cibercriminosos também avançaram em novas formas de ofuscar código e escapar da detecção de antivírus.

Hoje pesquisadores de segurança conseguem definir gerações de *Malware* nessa linha do tempo em que cada geração é marcada por um período conjunto de vulnerabilidades/ataques explorados e as medidas de defesa tomada, conforme verifica-se [Mil13]:

- A primeira geração de Malware (1986 – 1995) é marcada pelo proliferamento dos computadores pessoais (PC), aparecimento dos primeiros Vírus DoS de computador, surgimento do termo *hacker* para designar desenvolvedores de software malicioso e surgimento de companhias dedicadas à produção de antivírus comerciais. Os *hackers*<sup>1</sup> daquela época eram só pessoas curiosas com intenção de se divertir e quebrar sistemas;
- A segunda geração de Malware (1995 - 2000) é marcada pelo avanços da comunicação e *internet*, criação de sistema online de governo e negócios, surgimento de *Malware* voláteis e contagiosos e criação do firewall. Com os adventos da *internet* hackers começaram a criar grupos organizados, saindo do contexto de diversão e iniciando o cibercrime;
- A terceira geração de Malware (2000 - 2005) é marcada por ataques a vulnerabilidades de vários sistemas, uso intensivo de Worms, ataques por e-mail, e o surgimento de definição de vulnerabilidade em segurança como uma falha ou fraqueza de um componente de um sistema que pode ser tomado proveito. Para combater esse cenário surgiu os sistemas de prevenção de intrusão que é uma forma de proteger a rede contra ameaças identificadas; e
- A quarta geração de Malware (2005 - Hoje) é marcada pela mudança da mentalidade dos grupos de hackers antes eles só faziam para ganhar popularidade e se divertir agora eles produzem *Malware* com fins monetários. Crescimento intensivo do cybercrime, fraudes, tipos de *Malware* e *Malwares de Android*. Para combater esse cenário surgiu ferramenta para prevenção de ameaças em Android, sandbox e ferramentas de inteligência.

---

<sup>1</sup> **Hacker** é uma palavra em inglês do âmbito da informática que indica uma pessoa que possui interesse e um bom conhecimento nessa área, sendo capaz de fazer hack (uma modificação) em algum sistema informático. Muitas das atividades dos *hackers* são ilícitas ,..., pessoa que usa os seus conhecimentos de informática e programação para praticar atos ilegais são conhecidas como *crackers* e não *hackers* [hac].

### 2.3.6 ANÁLISE DE *MALWARES*

A principal atividade dos pesquisadores de *Malware* é a análise do software malicioso que corresponde a um processo manual em que se utiliza ferramentas para examinar os arquivos que compõem o código malicioso e o comportamento do arquivo quando executado ou instalado. A análise geralmente é feita por duas razões, pela resposta a incidentes e investigação. Respostas a incidentes acontecem quando um sistema foi atacado e os pesquisadores buscam informações sobre o incidente e como é a gravidade do ataque, como ocorreu a invasão, dados expostos ou vazados e como contê-lo. Outra razão é em busca de inteligência em que a empresa utiliza coletores para a captura um *Malware* destinado a ela e investiga quais vulnerabilidades essa ameaça está usufruindo, mapeando essa ameaça, seu potencial impacto e como ela pretendia invadir o sistema [DDTV<sup>+</sup>17].

Na análise de *Malware* existem duas formas de análise: Estática e Dinâmica. Análise estática é o processo em que se examina o software malicioso sem executá-lo através de ferramentas para exame da execução do código, descobrindo o *hash file* e fazendo uma engenharia reversa do código. Na análise dinâmica o processo envolve examinar o comportamento do *Malware* instalado e executar o código malicioso em um ambiente controlado e isolado, geralmente em uma máquina virtual, dessa forma minimiza-se os riscos para o analista [DDTV<sup>+</sup>17].

Quando comparamos as duas análises, a estática é mais rápida, porém não provê grande quantidade de detalhes sobre o *Malware*, sendo a dinâmica mais demorada, por causa da criação de um ambiente isolado e para execução do software malicioso, porém é mais detalhada e resistente a táticas de ofuscação de código. É importante lembrar que essas duas formas de análise retornam resultados diferentes e cada uma possui suas vantagens e desvantagens conforme comentado [DDTV<sup>+</sup>17].

### 2.4 DETECTORES DE *MALWARES*

Agora explica-se os conceitos chave para compreender como funciona um detector de *Malware*, sua definição, os métodos e técnicas que são utilizadas para construir um detector eficiente, as vantagens e limitações que cada técnica possui e as soluções disponíveis e mais utilizadas na atualidade.

#### 2.4.1 DETECÇÃO DE MALWARES

Para melhorar as técnicas tradicionais de análise de *Malware* e lidar com a grande quantidade e complexidade desses programas maliciosos, pesquisadores e companhias propuseram automatizar o processo de análise softwares maliciosos através de detectores. Um detector de *Malware* é programa que define se um software é maliciosos ou benigno. Um exemplo clássico de detector de *Malware* são os antivírus comerciais que geralmente possuem duas formas de análise: análise contínua do sistema em que periodicamente o programa escaneia o sistema removendo arquivos suspeitos e a análise pontual em que esse impede a instalação ou execução de um arquivo suspeito. Hoje, antivírus ficaram para trás porque não conseguem mais classificar *Malware* como antigamente, pois na atualidade vários desenvolvedores de *software maliciosos* aprenderam como escapar de antivírus comerciais [KJP19].

#### 2.4.2 TIPOS DE DETECCÃO DE MALWARES

*Malwares* são softwares que estão em constante mutação, pois os cibercriminosos sempre acham novas formas de penetração, novas vulnerabilidades dos sistemas e formas de passar por detectores. Sendo assim, durante esse processo de competição entre pesquisadores e cibercriminoso foram propostas várias técnicas para detectar se um aplicativo é ou não malicioso segundo a figura 2.7 abaixo e segundoo passaremos a expor [ASKA16].

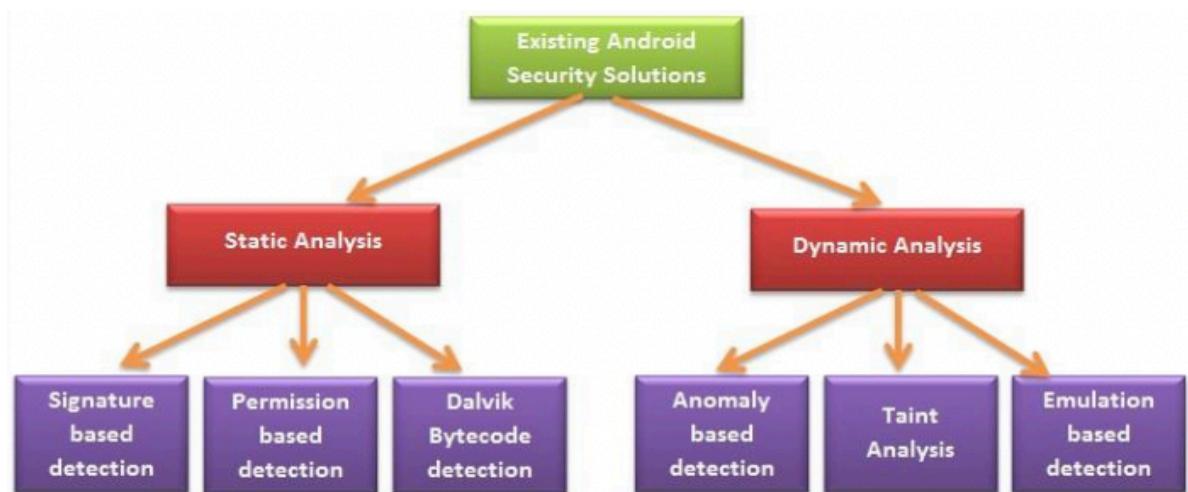


Figura 2.7: taxonomia de detectores de *Malware Android* existentes. Fonte: [ASKA16]

#### 2.4.2.1 SIGNATURE BASED APPROACH

É uma técnica de detector simples e bem efetiva para detectar *Malware* conhecidos. Ela foi uma das primeiras técnicas a surgir e geralmente é utilizada por antivírus comerciais. A detecção funciona da seguinte forma: o aplicativo é analisado estaticamente e é gerado uma assinatura única, extraída dos padrões semânticos do código, sendo depois comparada com a assinatura do banco de assinaturas de famílias de *Malware* conhecidas. Se a assinatura do aplicativo suspeito estiver no banco ele é classificado como malicioso. A limitação dessa técnica é que ela só é capaz de detectar famílias de *Malware* que são conhecidos e foram mapeados para o banco de dados do detector sendo assim, ela não detecta *Malwares* novos é dependente de um banco de dados atualizado [SS14].

#### 2.4.2.2 PERMISSION BASED DETECTION

É uma técnica de detector que utiliza os princípios de segurança de permissões do *Android* para detectar se um aplicativo é malicioso a partir das permissões que ele solicita. A detecção funciona através de uma análise estática no arquivo manifesto procurando um padrão entre os conjuntos de permissões de software benignos e o conjunto permissões de *Malware*. A limitação de um detector baseado em permissão é o fato que só é analisado o arquivo manifesto deixando os outros arquivos inexplorados que podem conter conteúdos maliciosos [SS14].

#### 2.4.2.3 DALVIK BYTECODE ANALYSIS

É uma técnica de detector que utiliza análise estática no aplicativo em que é percorrido o código no formato *dalvik bytecode* descobrindo o comportamento do arquivo e o fluxo de variáveis através de uma execução simbólica do código, obtendo assim, informação sobre as intenções do aplicativo e identificando se o arquivo pertence a uma família de *Malware*. A limitação dessa técnica é o grande uso de memória e de processamento para analisar o código no nível de instruções [ASKA16].

#### 2.4.2.4 ANOMALY BASED DETECTION

É uma técnica de detector que utiliza métodos da detecção de anomalia de ciência de dados junto com aprendizagem de máquina para definir se um aplicativo está com intuitos maliciosos. Para isso, o detector analisa diversos arquivos de sua escolha de forma dinâmica e caso o aplicativo fuja do comportamento normal de um aplicativo benigno, ele é classificado

como *Malware*. A limitação dessa técnica é a grande quantidade de falso positivo que é quando aplicativos benignos se comportam como *Malwares* [ASKA16].

#### 2.4.2.5 EMULATION BASED DETECTION

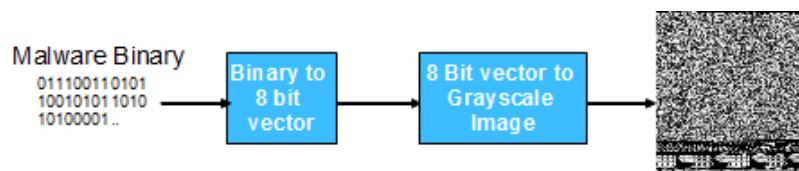
É uma técnica de detecção que utiliza um emulador para executar e analisar o aplicativo de uma forma dinâmica, dessa forma ele consegue executar todas as instruções do código de forma segura coletando todas informações do aplicativo e passando-as para uma análise heurística para detectar comportamento malicioso. De todas as formas de detectar *Malwares*, a baseada em emulação utiliza mais recursos do dispositivo *Android* [ASKA16].

#### 2.4.2.6 TAINT ANALYSIS

É uma técnica de detecção que faz análise dinâmica de software e utiliza *taint analysis* no código para rotular e rastrear dados sensíveis do dispositivo na execução do programa. Assim, mapeia-se o fluxo de informação dessas variáveis no código, para depois extrair o comportamento e comparar com o comportamento de aplicativo benignos classificando, concluindo se o software analisado era malicioso ou não. Porem, uma limitação dessa técnica é o tempo que custa para rodar e analisar aplicativo que usam essa técnica de detecção [ZAYC17].

#### 2.4.2.7 OUTRAS FORMAS

Surgiram várias novas formas de tentar classificar se um software *Android* é malicioso ou não, porém elas ainda estão em testes e não são muito utilizadas no mercado. Dentre as novas metodologias tem-se análises híbridas que corresponde a combinação de diferentes técnicas de análise e a utilização de votação no fim do processo. Outra forma nova é a transformação do código binário do aplicativo em imagens *greyscale* como mostra a Figura 2.8 abaixo, que utiliza essas bases de imagens junto com técnicas de classificação de imagem para detecção de *Malware* [ASKA16].



**Figura 2.8:** Transformação de *malwares* em imagens *greyscale*

#### 2.4.3 ONLINE MALWARE DETECTORS

Atualmente apareceu uma nova solução que tomou conta do mercado de detecção de software maliciosos que são os detectores online de *Malware* e seus serviços. Esses detectores consistem em um scanner online de programas maliciosos que são softwares na web, projetados para escanear o arquivo enviado pelo usuário utilizando diversas ferramentas para classificar o arquivo e depois mostrar um relatório completo do arquivo analisado. Hoje temos diversas opções de detectores online de *Malware*, cada um com suas vantagens e limitações. Dentre eles os mais famosos são: *VirusTotal*, *Hybrid analysis*, *MetaDefender*, *Herdprotect*, *Malwr*, entre outros.

Dentre os detectores de *Malware* online os mais famosos e mais utilizado entre empresas e pesquisadores é o *VirusTotal* que é serviço online, gratuito, de análise de arquivos e *websites* suspeitos. O *VirusTotal* foi projetado para detectar arquivos maliciosos e sites maliciosos (*Phishings*) através de escaneamento de arquivos e URLs respectivamente. Esses escaneamentos são feito a partir da análise de 70 antivírus parceiros e um sistema avançado de ferramentas de análise de URL e *blacklist* de domínios. Quando um usuário faz um upload de um arquivo no site, o *VirusTotal* checa se a assinatura do arquivo está presente no seu banco de dados, se estiver presente, o site retorna o relatório do arquivo, caso não esteja o *VirusTotal* utiliza o API dos antivírus parceiros e faz uma análise em cada um deles. No fim do processo, o *VirusTotal* retorna um relatório das análises e a classificação do software suspeito [virb].

O *VirusTotal* é uma ferramenta que foi desenvolvida pela *Hispasec* e depois comprada pela *Google* em 2012, se tornando uma ferramenta popular, eficiente e principalmente rápida em comparação com as outras soluções, graças a infraestrutura do *Google*. A ferramenta possui diversos serviços e uma comunidade de analistas de *Malwares* ativa, porém o site tem limite de tamanho de arquivos que podem ser analisados e também é dependente da qualidade de análises de seus parceiros. Pode-se ver um exemplo do seu funcionamento na Figura 2.9. [PYSW19].

The screenshot shows the VirusTotal interface for a file named 'myfile.exe'. A large red circle on the left indicates that 38 engines have detected this file. The main panel displays the file's details: size (3.50 KB), date (2019-10-09 06:47:23 UTC, 20 days ago), and a PDF icon. Below this, a table lists the detection results from various engines:

DETECTION	DETAILS	COMMUNITY	
Ad-Aware	① Exploit.PDF-Payload.Gen	AegisLab	① Hacktool.Win32.Pidief.31c
AhnLab-V3	① PDF/Exploit	ALYac	① Exploit.PDFDownloader
Arcabit	① Exploit.CVE-2009-0927.Gen, PDF:Ex...	Avast	① JS:Pdfka-gen [Expl]
AVG	① JS:Pdfka-gen [Expl]	Baidu	① PDF.Exploit.pidief.al
BitDefender	① Exploit.PDF-Payload.Gen	ClamAV	① Pdf.Dropper.Agent-1845292
Comodo	① Malware@mfeygrog4br	Cyren	① JS/Crypted.EH
Emsisoft	① Exploit.PDF-Payload.Gen (B)	eScan	① Exploit.PDF-Payload.Gen

**Figura 2.9:** Um Exemplo de como funciona o detector *VirusTotal* quando detecta um aplicativo malicioso e o relatório que ele retorna.

#### 2.4.4 APRENDIZAGEM DE MÁQUINA PARA DETECÇÃO DE MALWARES ANDROID

Existem várias formas de utilizar aprendizagem de máquina para detecção de *Malware* em *Android*. As duas formas mais notáveis, dentre os tipos de análise estática ou dinâmica, são respectivamente: A baseada em Permissões e a baseada em Anomalia. Como os experimentos desta monografia se baseiam em detectores criados a partir da extração dos recursos do arquivo manifesto, vai-se focar em artigos que utilizam aprendizagem de máquina na construção de detectores baseada em permissões.

Dentre os vários artigos existentes, encontrados e analisados sobre aprendizagem de máquina, escolhe-se os citados abaixo para uso no experimento pela sua disseminação, pela facilidade do uso e pela familiaridade.

- **Significant Permission IDentification (SIGPID)** é um detector de Malware baseado em permissão que utiliza apenas uma pequena parcela de permissões para classificar se um software é benigno ou maligno. Esse detector utiliza SVM como algoritmo classificador e tem como objetivo estudar a redução da quantidade de permissões utilizadas para melhorar os detectores. Também foi utilizado *Random Forest* para validação do detector [LSY+18];

- ***Permission-Induced Risk*** é outro detector em que é calculado o risco potencial de todas as permissões, depois essas permissões são classificadas em subconjuntos de acordo com seu risco e em seguida é aplicado algoritmos de aprendizagem de máquina em cima dessas permissões. Sendo o SVM, *Random Forest* e árvore de decisão aqueles utilizados para detectar aplicativos maliciosos [WWF+14];
- ***Mining API Calls and Permissions for Android Malware Detection*** que é um detector que utiliza outros recursos além de permissões do manifesto para executar a análise estática. Esse detector de *Malware* extrai permissões e chamadas de API relevantes dos aplicativos e logo após, constrói e avalia modelos criados a partir de KNN e *Naive bayes* conseguindo uma alta porcentagem de precisão [SD14]; e
- ***Permission-Based Feature Scaling Method for Lightweight Android Malware Detection***, que é um projeto que se foca mais em tornar a detecção mais rápida e mais leve para os processadores dos dispositivos *Android*. Funciona executando um processo de seleção de permissões para agilizar a execução de modelos de detecção sem prejudicar a precisão. Os modelos criados no projeto utilizam *Naive Bayes*, *SVM*, *Random Forest*, Árvores de Decisão [ZX19].

## 2.5 ESTADO DA ARTE PARA DETECÇÃO AUTOMATIZADA DE *MALWARE*

A área de detecção automatizada de *Malware* passou por diversos avanços desde o seu início com os antivírus tradicionais, que utilizavam assinaturas para detectar *Malwares* conhecidos, evoluindo atualmente para os detectores híbridos que combinam diversas técnicas complexas para detectar anomalias no sistema. Dentre os principais fatores que moldaram a evolução da área têm-se: os avanços na tecnologia e o redirecionamento do foco para dispositivos móveis. Nesta seção, vai-se pontuar alguns estudos notáveis sobre detecção utilizando análise estática e dinâmica para compreender o estado da arte em detecção de *Malwares* em *Android* [HAM19].

### 2.5.1 SOLUÇÕES COM ANÁLISE ESTÁTICA

As soluções de detecção de *Malware* com foco em análise estática podem utilizar três abordagens: Baseada em Assinaturas, Permissões e Análise de *Dalvik Bytecode*, conforme se segue:

- **Baseadas em Assinaturas:** *AndroSimilar* é uma solução que utiliza técnicas de “fuzzy hashing” para criar assinaturas e compará-las com sua grande base de dados de *Malwares Android*. Outra solução é *DroidAnalytics* que cria assinaturas através das chamadas de API do aplicativo e depois as correlaciona com as assinaturas do seu banco de dados através de pontuação;
- **Baseada em Permissões:** PUMA é um detector de *Malware Android* que retira permissões e “*API calls*” através de ferramentas e as compara e classifica entre aplicativos malicioso ou benigno através de *Machine Learning*. Outra análise proposta por *R.Sato* é análise através do arquivo manifesto em que é associado uma nota de malícia para o arquivo. Se este arquivo ultrapassar a nota limite é classificado como malicioso. Ainda, similar a esse último método é usado a regras de segurança pelo KIRIN em que é definido um conjunto de regras para aplicativos benignos seguirem, se as regras forem infringidas o aplicativo é classificado como malicioso; e
- **Baseada em Dalvik Bytecode:** *Karlsen* foi um dos primeiros a utilizar *Dalvik Bytecode* para detecção de *Malware*, no seu modelo era possível analisar o fluxo de dados e API sensíveis para ajudar na classificação dos aplicativos. Outro detector interessante é o SCANDAL que utiliza uma abordagem diferente em que os aplicativos são classificados como malicioso pela constatação de vazamentos de informações sensíveis [HAM19].

### 2.5.2 SOLUÇÕES COM ANÁLISE DINÂMICA

As soluções propostas envolvendo Análise Dinâmica de detecção de arquivos maliciosos para *Android* podem ser feitas em três abordagens: Baseado em Anomalia, utilizando *Taint Analysis* e por Emulação, conforme se segue:

- **Baseada em Anomalias:** *CrowDroid* é um detector que faz rastreamento de chamadas do sistema e cria *logs* para detectar anomalias, considerando que aplicativos maliciosos fazem mais chamadas que envolvem dados sensíveis. Uma abordagem interessante que utiliza aprendizagem de máquina é o *Andromly* que utiliza algoritmos para, constantemente, monitorar o estado do dispositivo, conseguindo assim, diferenciar e classificar o comportamento maligno do benigno;

- **Baseado em Taint Analysis:** *TaintDroid* é talvez a solução mais famosa que utiliza esse método, pois automaticamente rotula os dados sensíveis e os rastreia dentro do dispositivo registrando suas atividades em documento para futura análise, permitindo classificar as intenções dos aplicativos; e
- **Baseado em Emulação:** *DroidScope* é uma solução detecção que utiliza método de emulação em que o detector cria um ambiente seguro e contido (*sandbox*) onde o aplicativo é executado e é monitorado provendo o rastreamento do estado atual do sistema operacional e do Dalvik fazendo uma introspecção da máquina virtual dessa forma classificou as intenções do aplicativo na sandbox [HAM19].

### 3. EXPERIMENTO

Este capítulo detalha todos os passos para execução do experimento. Com o objetivo de analisar a importância e influência da escolha de atributos para detectores de *Malwares* de smartphones com sistemas *Android*, serão desenvolvidos modelos de aprendizagem de máquina que utilizaram informações adquiridas através de análise estática de arquivos manifestos dos arquivos “APK”. Aqui detalhamos as bases disponíveis de dados online, em que faremos uma extração bruta dos dados coletados dessas bases, logo após faremos um pré processado dos dados para depois executar e criar modelos de aprendizagem de máquina para coletar os resultados. Através desses experimentos conseguiremos comprovar quais *features* extraídas tem mais relevância para um detector baseado em permissão. Em outras palavras, deseja-se ao final comprovar se existe viabilidade da utilização de “*intents*” como “*features*” em detectores de *Malware Android* baseado em permissão.

#### 3.1 BASES DE DADOS

Antes de entrar em detalhes de quais bases estão disponíveis hoje na internet e suas alternativas, precisa-se primeiro entender a seriedade da disseminação e análise desses arquivos perigosos. Quando se encontra uma base de dados e/ou datasets de “*Live Malware*” se entende que essas bases possuem os arquivos executáveis do *Malware* “EXE” ou “APK”, que quando executados até acidentalmente irão infectar a máquina do usuário e por isso é necessário uso de uma máquina virtual. Além disso, esses artefatos podem ser utilizados para fins negativos. Tendo esse fato em mente, a maioria das grandes bases de *Live Malware* disponíveis na internet tem uma extensa documentação sobre a sua política de distribuição e utilização. Dessa forma, alerta-se ao leitor que antes de fazer requisição para o uso desses sites leia a sua política e também prepare um ambiente seguro para não infectar a sua máquina.

##### 3.1.1 CLASSIFICAÇÃO DAS BASES

Quando se analisa as bases de dados disponíveis com relação a suas políticas e custo do serviço pode-se classificá-las em três tipos: Bases Públicas, Bases Pagas e Bases Gratuitas. Bases públicas são aquelas que estão em sites que fazem a distribuição completamente gratuita e sem política rígida de sua utilização e disseminação. Como exemplos dessas bases tem-se: Repositórios Github, UNB e Contágio Dump. Já as bases pagas são bases distribuídas por sites em que o usuário paga, geralmente por mês, aos donos do servidor para ter acesso aos arquivos

executáveis. Como exemplo tem-se: *Virussign*. As bases públicas necessitam de registro, correspondem a bases com uma grande política de distribuição dos arquivos e necessitam passar por uma aprovação do site para ter acesso aos *Malwares*. Essas bases geralmente são utilizadas por pesquisadores e entusiastas da área de análise de *Malware*. Para ter acesso aos arquivos maliciosos o usuário necessita primeiro fazer uma requisição aos donos do site, explicando a necessidade e o motivo para ter acesso aos arquivos executáveis. Além disso, os sites com essas bases exigem ao solicitante possuir alguma afiliação com um órgão de pesquisa como universidades, laboratórios ou empresa de segurança. Exemplos de bases assim tem-se: *VirusShare*, *Androzoo*, *Drebin* e *MalwareShare*.

Para facilitar a visualização das bases disponíveis foi criada a Tabela 3.1 abaixo, que mostra algumas das bases mais famosas exibindo nome, tipo, sistema operacional e URL para acesso do website. É importante lembrar que alguns URLs podem ficar fora do ar, principalmente os URLs de bases públicas por causa da gravidade dos arquivos distribuídos.

**Tabela 3.1:** Bases Disponíveis Online (Autor).

NOME	Tipo	Sistema operacional	URL
CONTAGIO	Pública	Android	<a href="http://contagiodump.blogspot.com/">http://contagiodump.blogspot.com/</a>
SYRIAN MALWARE	Pública	Windows	<a href="http://syrianmalware.com">syrianmalware.com</a>
ANDROZOO	Gratuita	Android	<a href="https://androzoo.uni.lu/">https://androzoo.uni.lu/</a>
THEZOO	Pública	Android / Windows	<a href="https://github.com/ytisf/theZoo">https://github.com/ytisf/theZoo</a>
android-malware (Github)	Pública	Android	<a href="https://github.com/ashishb/android-malware">https://github.com/ashishb/android-malware</a>
VIRUSSHARE	Gratuita	Android / Windows	<a href="https://virusshare.com/">https://virusshare.com/</a>
VIRUSSIGN	Paga	Android / Windows	<a href="http://www.virussign.com/">http://www.virussign.com/</a>
UNB	Pública	Android	<a href="https://www.unb.ca/cic/datasets/invesandmal2019.html">https://www.unb.ca/cic/datasets/invesandmal2019.html</a>
MALSHARE	Gratuita	Android / Windows	<a href="https://malshare.com/">https://malshare.com/</a>
VX VAULT	Gratuita	Windows	<a href="http://vxvault.net">http://vxvault.net</a>
KERNELMODE	Gratuita	Android / Windows	<a href="http://kernelmode.info">kernelmode.info</a>
AMD	Gratuita	Android	<a href="http://amd.arguslab.org/">http://amd.arguslab.org/</a>
MALWARE DOMAIN LIST	Pública	Windows	<a href="http://malwaredomainlist.com">malwaredomainlist.com</a>
VIRUSTOTAL	Gratuita	Android / Windows	<a href="http://virustotal.com">virustotal.com</a>
HYBRID ANALYSIS	Gratuita	Android / Windows	<a href="http://hybrid-analysis.com">hybrid-analysis.com</a>
KOODOUS	Gratuita	Android / Windows	<a href="http://koodous.com">koodous.com</a>
DREBIN	Gratuita	Android	<a href="https://www.sec.cs.tu-bs.de/~danarp/drebin/">https://www.sec.cs.tu-bs.de/~danarp/drebin/</a>
GENOME PROJECT	Gratuita	Android	<a href="http://www.malgenomeproject.org/">http://www.malgenomeproject.org/</a>
CONTAGIO MOBILE	Pública	Android	<a href="http://contagiominidump.blogspot.com/">http://contagiominidump.blogspot.com/</a>
FREE MALWARE SAMPLES	Pública	Android	<a href="https://hseoeonr.tk/health-fitness/android-malware-samples.php">https://hseoeonr.tk/health-fitness/android-malware-samples.php</a>
DAS MALWERK	Pública	Windows	<a href="https://dasmalwerk.eu/">https://dasmalwerk.eu/</a>
AVCEASAR	Gratuita	Windows	<a href="http://malware.lu">malware.lu</a>

Para o experimento foram escolhidas 3 bases de aplicativos maliciosos e uma base de aplicativos benignos. Nas bases maliciosos foram escolhidos duas que necessitam registro por causa da alta quantidade e variedade de aplicativos e uma pública para facilitar a reprodução dos experimentos. Por causa da dificuldade de encontrar bases benignas de aplicativos foi escolhida apenas uma base com a finalidade de facilitar a reprodução dos experimentos.

### 3.1.2 BASES DE *MALWARES ANDROID* SELECIONADAS

VirusShare é um repositório de diversos tipos de *Malwares* projetado para proporcionar aos pesquisadores e entusiastas da área acesso a uma vasta gama de dados de *live malware* para projetos de segurança. A base do VirusShare foi a primeira base de aplicativos maliciosos que consegui ter acesso. Para conseguir acesso a essa base foi feito uma solicitação por e-mail aos administradores do site explicando o motivo e o objetivo do trabalho, sendo prontamente atendido com um *login* e senha para ter acesso aos downloads e torrents dos arquivos executáveis [vira].

A *Androzoo* é um repositório de aplicativos maliciosos *Android* administrado pela universidade de Luxemburgo que coleta *Malwares Android* de diversas fontes, incluindo até lojas oficiais de aplicativos como *GooglePlay*. O *Androzoo* foi feito com o intuito de proporcionar um conjunto de vastos dados para contribuir com pesquisas em andamento e possibilitar novos tópicos relacionados com *Malwares Andriod*. Igualmente ao *VirusShares* foi feito um solicitação para os administradores do site explicando a motivação e objetivo de pesquisa que nesse caso deve, obrigatoriamente, ter uma afiliação com uma instituição de pesquisa, como universidades ou empresas de segurança digital [ABKLT16].

Após a solicitação os administradores retornam com um *API key* para ser utilizado na escolha de arquivos APK que deseja obter da base. A AndroZoo aconselha o uso de um script ‘az’ criado pelo *Artem Kushnerov* para facilitar o download de grande número de *Malwares*. No Github do criador (<https://github.com/ArtemKushnerov/az>) possui uma vasta documentação do script ‘az’, em que se ensina como fazer comandos complexos e utilizar a *API key* da melhor forma possível. Um exemplo de comando é dado na Figura 3.1 [scr].

```
az -n 10 -d 2015-12-11: -s :3000000 -m play.google.com,appchina
```

**Figura 3.1:** Exemplo de comando utilizando script az em que é solicitado 10 aplicativos iniciando da data 11/12/2015 de tamanho de até 3.000.000 bytes que foram publicados nos mercados play.google.com ou appchina. Fonte: [scr]

*Android-Malware GitHub* é um repositório do *GitHub* que contém uma série de arquivos .APK organizados em pasta de acordo com a família de *Malwares* de cada arquivo executável. Essa base tem um pequeno número de *Malwares*, porém se tratando de um

repositório do GitHub, ela é gráts e simples para baixar, sem restrições, como as outras bases encontradas na internet[mal].

*Android Adware and General Malware Dataset* (UNB) é um repositório gratuito de *Malwares* que contém aplicativos maliciosos e aplicativos benignos. A base de dados é administrada pela universidade de *New Brunswick* no Canadá e não possui restrições sobre o downloads aos datasets de *Malwares* ou datasets benignos do site. Dessa forma, dentro do repositório gerenciado pela universidade encontra-se bases separadas de aplicativos benignos. Essa base será utilizada no projeto denominada da base benigna na pesquisa de *Adware Androi*. Um fato interessante é que a origem dos arquivos da base benigna são aplicativos gratuitos populares da GooglePlay coletados no período 2015 até 2017 [LKG<sup>+</sup>17].

### 3.1.3 BASE DE DADOS DE DADOS BRUTOS

A base de dados brutos do projeto é composta por cinco bases de dados, sendo três maliciosas e duas benigna. Tendo em vista as limitações de tempo da monografia e do tamanho da base benigna utilizada no projeto foi delimitado um limite de no máximo 1.700 de aplicativos para as bases maiores e 600 aplicativos para as menores. O conteúdo de cada base é definido na Figura 3.2:

**Tabela 3.2:** Bases de dados brutos (Autor)

BASE	MALICIOSIDADE	QUANTIDADE	ORIGEM	PERÍODO
GitHub	MALIGNA	293	Variada	2015 - 2019
VirusShare	MALIGNA	1564	Variada	2017
AndroZoo	MALIGNA	1607	GooglePlay	2016 - 2019
UNB	BENIGNA	584	GooglePlay	2017
UNB	BENIGNA	1615	GooglePlay	2015 - 2017

### 3.2 EXTRAÇÃO DE FEATURES

O próximo passo foi o download das bases com os arquivos executáveis dos *Malwares* e a extração de *features* relevantes desses arquivos. Para escolher que *features* que se deseja extrair precisa-se primeiro selecionar como o detector irá fazer a detecção dos arquivos maliciosos. Tendo em vista os limites de tempo decidi usar a metodologia da detecção estática baseada em permissão mesmo que essa forma tenha um alto índice de falso positivo comparado

com as outras metodologias, pois ainda é a forma mais rápida e com o processamento mais leve entre as alternativas disponíveis.

### 3.2.1 METODOLOGIA DE EXTRAÇÃO

Depois de decidir a forma que o detector de *Malwares* irá funcionar, foca-se agora em como fazer a extração das *features*, relembrando o que foi mostrado na seção de tipos de detector de *Malwares Android*, que um detector baseado em permissão deve buscar padrões dentro do arquivo manifesto e descobrir as intenções da aplicação. Tendo isso em mente, busquei pela internet formas de “*parser*<sup>2</sup>” para o arquivo manifesto e bibliotecas que podiam ser utilizadas para diminuir o retrabalho das atividades.

A solução encontrada foi utilizar uma biblioteca na linguagem “*Python*” chamada *AndroGuard* que é um conjunto de ferramentas criada com o propósito de investigar e explorar aplicativos de forma estática. *Androguard* é composto por inúmeras ferramentas dentre elas temos: um decoder dos recursos da aplicação, um descompilador do APK, criação gráficos de controle de fluxo, ferramentas de engenharia reversa, explorador de certificados, além de um *parser* do *Android-Manifest.xml* [D+13].

Para a extração de features do manifesto dos aplicativos malicioso e benignos foi criado um código *python* que percorre uma lista contendo endereços dos arquivos APK de uma base e em seguida, faz a extração de permissões dos aplicativos através da biblioteca *Androguard* em seguida retorna um arquivo CSV com as permissões contidas dentro da base analisada. Também foi criado um código *python* para dar suporte a extração de features. Nesse código é feita a criação de uma lista com os endereços dos arquivos executados dado o endereço da base.

---

<sup>2</sup> **Parser** é um programa ou parte de um programa que interpreta a entrada para um computador reconhecendo palavras-chave ou analisando a estrutura da sentença. [par]

### 3.2.2 CÓDIGO DE EXTRAÇÃO

Agora olha-se com mais detalhes para o funcionamento dos dois códigos criados: *Extractor.py* que é responsável pela extração de permissões dos aplicativos e *MakeCSV.py* que é o código suporte que transforma a base em um CSV para ser lido pelo *Extractor.py*.

```

import os
import csv

# Find Malwares in the Document
mydir = "D:\\Androzoo"

x = os.walk(mydir)
|
# Start MAKE CSV with malware.apk dirs
print("START")
with open('Malware_dir_Androzoo.csv','w', newline='') as csv_file:
    fieldname = ['MD', 'NUMBER']
    writer = csv.DictWriter(csv_file, fieldnames = fieldname)

    malware_Count = 0
    for root_dir, sub_dir, files in x:
        print ("ROOT DIR")
        print (root_dir)
        #print ("SUB DIR")
        #print (sub_dir)
        #print ("FILES")
        #print (files)
        print ("MALWARES")
        for item in files:
            if ".apk" in item:
                malware_Count = malware_Count + 1
                name = root_dir + "\\\" + item
                name = name.replace('\\','\\\\')
                print (name)
                writer.writerow({'MD' : name , 'NUMBER' : malware_Count})
    print ("=====")

```

**Figura 3.2:** Código suporte MakeCSV.py.

*MakeCSV.py* representado pela Figura 3.2 é um código *python* que foi feito com o objetivo criar um arquivo legível que consiga referenciar arquivos executáveis dentro de uma base. O código *python* funciona da seguinte forma: o usuário dá como entrada o diretório da base e utiliza “*OS.Walk*” para percorrer os arquivos dentro da base buscando os arquivos que terminam com APK e os registra dentro de um arquivo CSV.

*Extractor.py* representado pela Figura 3.3 é um código *python* projetado para extrair as permissões dos arquivos *Manifest.xml* dos aplicativos de uma base. O código funciona da

seguinte forma: o usuário dá como entrada um arquivo .CSV contendo os caminhos para os arquivos executáveis da base, logo após o código percorre a base no formato CSV e utiliza o *Androguard* para analisar os arquivos .APK criando um objeto para cada arquivo, tornando fácil o acesso aos recursos do manifesto em que é armazenado os dados em um novo arquivo .CSV para ser utilizado por técnicas de aprendizagem de máquina.

```

import csv
import zipfile
import pandas as pd
from androguard.misc import AnalyzeAPK

# Select file
# Use skiprows to choose starting point and nrows to choose number of rows
data = pd.read_csv('VirusShare.csv', skiprows = 0, nrows=1600)

# Criar um CSV com permissões
csv_base = open('VirusShare_All.csv','w', newline='')
writer = csv.writer(csv_base)
writer.writerow(['Numero','Name','Permissions','Provider','Services','SINT','Receivers','RINT'])

# MY helper to extract intents
def helper(obj,itemtype):
    alista =[]
    for name in obj:
        for action,intent_name in a.get_intent_filters(itemtype, name).items():
            for intent in intent_name:
                alista.append(intent)
    return alista


i = 0
while i < 1600:
    file = (data.iloc[i][0])
    numero = (data.iloc[i][1])
    try:
        a, d, dx = AnalyzeAPK(file)
        print ("====> " + file + " <====")
        perms = (a.get_permissions())
        prov = (a.get_providers())
        serv = (a.get_services())
        sint = helper(serv,'service')
        receivers = a.get_receivers()
        r = []
        for x in receivers: r.append(x)
        rint = helper(receivers,'receiver')
        writer.writerow((numero,file,perms,prov,serv,sint,r,rint))
        csv_base.flush()
    except zipfile.BadZipFile: # WINDOWS Malware
        print ('Windows' + str(numero) )
    except KeyError: # DIC ERROR
        print('Error DIC =====>' + str(numero))
    except TypeError:
        print ('Error : Type Downloader =====>' + str(numero))

    print ("====> " + str(numero) + " <====")
    i = i + 1

```

**Figura 3.3:** Código *Extractor.py* que extrai os recursos dos arquivos APK.

### 3.2.3 RESULTADO DA EXTRAÇÃO

Após a execução do código *Extractor.py* nas bases baixadas tem-se quatro arquivos: *AndrozooBruto.csv*, *VirusShareBruto.csv*, *GithubBruto.csv* e *UNBBenignoBruto.csv*. Antes de passar para a fase de pré processamento é feito algumas mudanças nesses arquivos. Primeiro é juntado as bases de *Malwares* com a base benigna UNB, exceto a base GitHub que é concatenada apenas com uma parte da base UNB de aplicativos de 2017, depois é criado a coluna ‘*Malware*’ com o valores binários 0 e 1 correspondente a benigno e maligno, respectivamente. Em seguida são removidos ruídos e duplicatas das bases ao passar a base por uma função *Uppercase*. Além disso, é criado uma base ‘*ALLBruto.CSV*’ com todos os arquivos malignos e benignos. As bases resultantes estão representadas na Tabela 3.3.

**Tabela 3.3:** Bases de dados extraídos bruta (Autor)

BASE	BENIGNO - %	MALIGNO - %	QUANTIDADE DE APlicativos
<i>VirusShare.CSV</i>	50.8	49.2	877
<i>AndroZoo.CSV</i>	50.7	49.3	3222
<i>Github.CSV</i>	70.0	30.0	3179
<i>ALL.CSV</i>	32.2	67.8	5079

### 3.3 PRÉ PROCESSAMENTO DE DADOS

O próximo passo do projeto foi fazer o pré processamento dos dados obtidos através da extração das bases benignas e malignas. Para isso foi analisado brevemente os arquivos CSV gerados pela execução do código *Extractor.py* em todas as bases. Foram notado alguns detalhes que devem ser tratados nas bases, como: a necessidade da transformação de listas de recursos do manifesto em colunas e a remoção dos recursos duplicados. Em seguida, foi feito um novo conjunto de bases de dados a partir da base bruta no formato CSV em que cada um possui apenas um tipo ou combinação de recursos extraídos do arquivo manifesto, como exemplo: uma base com apenas permissões solicitadas dos aplicativos. Essas novas bases nos ajudaram e participaram no entendimento de quais são os melhores recursos a serem extraídos do arquivo manifesto. Para ter um pré processamento claro e efetivo foi utilizado “*Jupyter*<sup>3</sup>” uma

---

<sup>3</sup> *Jupyter* - é um projeto de código aberto sem fins lucrativos, nascido do Projeto IPython em 2014, oferece suporte à ciência de dados interativa e à computação científica em todas as linguagens de programação (<https://jupyter.org/>). [KRKP+16]

ferramenta *opensource* que auxilia nessa tarefa e “*Pandas*<sup>4</sup>” uma biblioteca *python* que facilita a manipulação dos dados.

```
In [1]: import pandas as pd
import ast

In [2]: # Read Database.csv
data = pd.read_csv('ALL_Untreated.csv', sep = ',')

In [3]: data.head()

Out[3]:
  Numero Malware          Name           PERMISSIONS           PROVIDER
0      2    1 D:\torrent\Malware [ANDROID.PERMISSION.READ_SYNC_SETTINGS, ...
1      3    1 D:\torrent\Malware           [ANDROID.PERMISSION.NFC, ...
2      4    1 D:\torrent\Malware [ANDROID.PERMISSION.READ_SYNC_SETTINGS, ...
3      5    1 D:\torrent\Malware [ANDROID.PERMISSION.CHANGE_CONFIGURATION, ...
4      6    1 D:\torrent\Malware [ANDROID.PERMISSION.SET_WALLPAPER, ...

In [4]: m = data.loc[data['Malware']==1]
m.shape

Out[4]: (3326, 9)

In [5]: b = data.loc[data['Malware']==0]
b.shape

Out[5]: (1588, 9)

In [12]: # SEE SHAPES
lista_dfs = [final,permissions_only,provider_only,services_only,sint_only,receivers_only,rint_only,pro_per_int_only,per_i
nt_only,per_pro_only,pro_int_only,int_only]

for i in lista_dfs:
    print(i.shape)

(4914, 29347)
(4877, 2566)
(1388, 1432)
(3241, 9472)
(1588, 2160)
(347, 3391)
(3089, 5415)
(4914, 11486)
(4914, 10055)
(4901, 3937)
(4523, 8981)
(4476, 7550)

In [13]: final.to_csv('AndroZoo_TREATED.csv')
pro_per_int_only.to_csv('pro_per_int_only.csv')
per_int_only.to_csv('per_int_only.csv')
per_pro_only.to_csv('per_pro_only.csv')
pro_int_only.to_csv('pro_int_only.csv')
int_only.to_csv('int_only.csv')
permissions_only.to_csv('permissions_only.csv')
provider_only.to_csv('provider_only.csv')
services_only.to_csv('services_only.csv')
sint_only.to_csv('sint_only.csv')
receivers_only.to_csv('receivers_only.csv')
rint_only.to_csv('rint_only.csv')
```

**Figura 3.4:** Código ‘*Pre processamento.ipynb*’, representado em corte início e fim (Autor). O código completo encontra-se no Anexo 1 deste trabalho.

Para realizar o pré processamento dos dados foi elaborado um código que funciona com todas as bases brutas “*Preprocessamento.ipynb*” escrito em *python* através da ferramenta “*jupyter notebook*”. O programa funciona da seguinte maneira: primeiro é importado a base não tratada em arquivo CSV para um *dataframe* dentro do código para facilitar o manuseio. Em seguida, é criadada a função “*Trans*” para auxiliar na transformação de colunas com listas para *dataframes* com colunas binárias, logo após, essa função é aplicada em todas as colunas com listas relacionadas ao recursos do manifesto formando novos *dataframes*.

---

<sup>4</sup> **Pandas** - é uma biblioteca de código aberto, licenciada por BSD, que fornece estruturas de dados de alto desempenho, fáceis de usar com ferramentas de análise de dados para a linguagem de programação Python. (<https://pandas.pydata.org/>) [Nel15]

Após a geração dos *dataframes* com os recursos separados consegue-se fazer novas bases de dados a partir das combinações dos novos *dataframes*, em que se pode testar, examinar, validar e criar novas hipóteses para o projeto. Em seguida são formados dois tipos de bases: a “*Mixed*” que é a combinação de bases que foram utilizadas para testar possíveis combinações de recursos extraídos do arquivo manifesto e a “*Alone*” que são bases que foram criadas a partir de apenas um recurso que mostra a relação desses recursos com maliciosidade de um aplicativo. Essas bases passam por um tratamento de dados em que são removidos dados nulos, substituídos valores “Nan” por “0” e removidos linhas que apenas tem valor “0”. Por fim, essas bases são transformadas em arquivos CSV para serem utilizados pelos próximos códigos.

**Tabela 3.4:** Bases de dados extraídos após pré processamento e tratamento dos dados (Autor)

TIPO	AndroZoo		GitHub		VirusShare		ALL	
	linhas	Col	linhas	Col	linhas	Col	linhas	Col
<b>TOTAL</b>	<b>3127</b>	<b>14752</b>	<b>831</b>	<b>6053</b>	<b>3125</b>	<b>26281</b>	<b>4914</b>	<b>29347</b>
Permissions_only	3098	1580	823	704	3110	2161	4877	2506
Provider_only	905	725	360	286	1149	1292	1388	1432
Services_only	1716	4766	562	1945	2494	8440	3241	9472
Sint_only	676	784	286	299	1337	1927	1525	2136
Receivers_only	1676	4016	576	1668	2454	7514	3147	8391
Rint_only	1633	2886	567	1153	2428	4952	3089	5415
Provider_Permissions_Intent	3127	5972	831	2442	3125	10329	4914	11486
Permissions_Intent	3127	5248	831	2154	3125	9038	4914	10055
Provider_Permissions	3114	2304	830	992	3112	3452	4901	3937
Provider_Intent	2736	4393	690	1739	2734	8169	4523	8981
Intent_only	2689	3669	665	1451	2687	6878	4476	7550

### 3.4 TÉCNICAS DE APRENDIZAGEM DE MÁQUINA

Após o pré processamento das bases antigas e geração de novas bases de dados utilizar-se, agora, técnicas de aprendizagem máquina em cima dos dados gerados para fazer constatações e validar hipóteses. Para isso, foi produzido um código *python* aliado a *jupyter notebook* ‘*AI.ipynb*’ que seja capaz de processar todos os tipos de bases e consiga gerar diversos resultados a partir de classificadores de aprendizagem de máquina definidos no código.

O código *AI.ipynb* é um código *python* que funciona com todas as bases. O código é dividido em duas partes: filtragem e classificadores. O programa inicia com a parte de filtragem que funciona da seguinte maneira: primeiramente é importado a base pré processada que se deseja analisar, depois gera-se um *dataframe* com os recursos a serem analisados e quantas vezes eles aparecem dentro da base CSV. Quando se analisa os recursos utilizados, percebe-se que mais de 90% são utilizados apenas uma única vez e nota-se que é grande uso de recursos

não oficiais. Então, para melhorar a nossas bases, passa-se essas primeiro, por um filtro em que são identificados os recursos utilizados em mais do que de cinco vezes e os recursos oficiais conhecidos.

Depois da filtragem foram feitos breves testes em que se percebeu um alto desbalanceamento das bases, então foi adicionado ao código uma parte para redimensionar a base utilizando o método *Undersampling*, que é uma forma de redimensionar bases, excluindo-se linhas de forma aleatória da base da classe majoritária até igualar a classe minoritária. Após esse tratamento, passa-se a aplicar as técnicas de aprendizagem de máquina.

```
In [1]: import pandas as pd
In [2]: # Adicionar Base CSV Pre processada desejada
df = pd.read_csv('provider_only.csv')
In [3]: df.head()
Out[3]:
   Unnamed: 0  Malware  ABBI.IO.ABBISDK.PROVIDERS.MULTIPROCESSPERFERENCESPROVIDER  ANDRO.PAKISTANTV.BD_PROVIDER  ANDROID.PPMEDIA.PROVI
0          1        1                           0.0                         0.0
1          9        1                           0.0                         0.0
2         10        1                           0.0                         0.0
3         15        1                           0.0                         0.0
4         18        1                           0.0                         0.0
5 rows × 1433 columns

In [4]: # Gerar Dataframe com Nomes e Vezes que o valor repetiu
x = df.sum()
l1 = x.to_list()
l2 = df.columns.tolist()
fd = pd.DataFrame(list(zip(l2, l1)),columns =['Name', 'val'])
In [5]: fd
Out[5]:
      Name    val
0  Unnamed: 0  3026270.0
1     Malware    650.0
2  ABBI.IO.ABBISDK.PROVIDERS.MULTIPROCESSPERFERENCE...    1.0
3     ANDRO.PAKISTANTV.BD_PROVIDER    1.0
4     ANDROID.PPMEDIA.PROVIDER.MEDIAPROVIDER    4.0
...
1428  TV.DANAMAKU.BILI.PROVIDERS.BILISTATUSPROVIDER    1.0
1429  VN.MECORP.IWIN.MYFILECONTENTPROVIDER    1.0
1430  WIKEM.CHRIS.WIKEMV3.DICTIONARYPROVIDER    1.0
1431  WSJ.UI.SEARCH.SEARCHSUGGESTIONPROVIDER    1.0
1432  YOUTUBE.IN.SPARK.ENERGY.EBPROVIDER    1.0
1433 rows × 2 columns

In [6]: fd.shape
Out[6]: (1433, 2)
In [7]: # Numero de recursos chamados mais de uma vez (MAIS DE 12000 recursos nao sao chamados)
umavez = fd.loc[fd['val'] > 1]
print("INFO Dataset: ", umavez.shape)
```

\

```
for train_i, test_i in folds.split(x,y):
    x_train, x_test, y_train, y_test = x[train_i], x[test_i], y[train_i], y[test_i]
    trash, Score = Classificador('bb',x_train,y_train,x_test,y_test)
    total = total + Score
    print("KFOLD %d : %f" % (valor,Score))
    valor = valor + 1
print("MEDIA DOS VALORES : " + str(total/5))

KFOLD 1 : 66.923077
KFOLD 2 : 75.000000
KFOLD 3 : 71.538462
KFOLD 4 : 71.923077
KFOLD 5 : 71.153846
MEDIA DOS VALORES : 71.3076923076923

In [36]: # KFOLD RANDOM F
resultados = []
total = 0
valor = 1

for train_i, test_i in folds.split(x,y):
    x_train, x_test, y_train, y_test = x[train_i], x[test_i], y[train_i], y[test_i]
    trash, Score = Classificador('rf',x_train,y_train,x_test,y_test)
    total = total + Score
    print("KFOLD %d : %f" % (valor,Score))
    valor = valor + 1
print("MEDIA DOS VALORES : " + str(total/5))

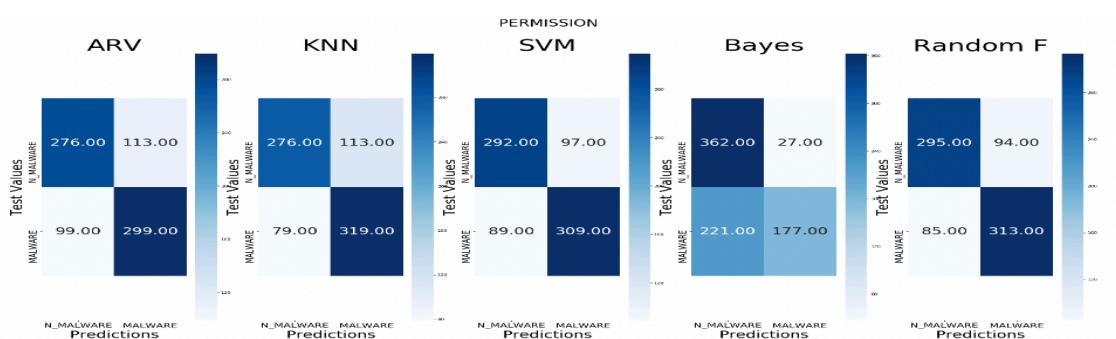
KFOLD 1 : 76.538462
KFOLD 2 : 78.046154
KFOLD 3 : 77.307692
KFOLD 4 : 78.846154
KFOLD 5 : 75.769231
MEDIA DOS VALORES : 77.46153846153847
```

**Figura 3.5:** Código ‘AI.ipynb’, representado em corte inicio e fim (Autor). O código completo encontra-se no Anexo 2 deste trabalho.

A segunda parte do programa inicia com a importação de bibliotecas de visualização de dados '*Matplotlib*' e '*Seaborn*' e bibliotecas de aprendizagem de máquina '*Scikit-learn*' e '*Sklearn*' para agilizar a aplicação dos algoritmos. Logo após, separa-se os valores que classificaram e o valor da classe dentro do *dataframe* tratado. Depois disso é feito o uso do '*train test split*' para separar a base treino e a base teste, utilizando nesse código a separação de 75% para a base treino e de 25% para base teste, apenas pelo fato de ser o valor padrão do "Sklearn". Em seguida é criado um função "Classificador" que organiza o uso de algoritmos classificadores em que essa função tem como entrada o tipo de classificador desejado e as bases treino e teste.

Em seguida, essa função é executada com 5 tipos de classificadores definidos na função (Árvore, *KNN*, *SVM*, *Naive Bayes* e *Random Forest*) e depois de coletados os resultados é feito uma matriz de confusão em cima dos resultados alcançados que é uma representação gráfica capaz de mostrar falso positivos e falsos negativos, dados cruciais para analisar detectores de *Malwares* conforme a Figura 3.4. Posteriormente, para fugir dos problemas de sorte na divisão da base treino e base teste, se faz a '*CrossValidation*' utilizando '*Stratified Kfold*' que é forma de fazer *Kfold* para problemas de classificação.

Ainda, no programa, foi testado qual seria a melhor quantidade de *folds* para serem utilizadas no *kfold*, definido-se 5 *Folds* no *Kfolds*, pois foi esse que apresentou melhores resultados durante os testes. Depois de definido executa-se o *Kfold* combinado com o classificador para avaliar os modelos de classificação empregados. É importante ressaltar que não houve um refinamento apropriado dos parâmetros dos algoritmos classificadores, como tamanho da árvore, melhor k para *nearst neighbors*, número de estimadores do *random forest*, dentre outros, pois o objetivo do trabalho é analisar a importância da seleção de *features* extraídas.



**Figura 3.6:** Exemplo de matriz de confusão gerada a partir do IA.ipynb

### 3.4.1 RESULTADOS

Para facilitar a visualização dos resultados apresenta-se quatro tabelas correspondente aos resultados da execução do *AI.ipynb* em cada uma das bases pré processadas. Cada tabela mostra um conjunto de valores da precisão dos algoritmos obtidos através da execução do código. Além disso, tem-se uma coluna e uma linha com as médias dos resultados encontrados para facilitar a comparação entre algoritmos. Porém, em se tratando de detectores automáticos de *Malwares*, não se pode avaliar um modelo de aprendizagem apenas através dos valores de precisão.

**Tabela 3.5:** Tabelas correspondente ao resultados da execução do *AI.ipynb* em cada uma das bases pré processadas.(Autor)

ANDROZOO - Precisão %						
TIPO	ARV	KNN	SVM	BAYES	R F	MÉDIA
<b>TODAS FEATURES</b>	72,70	72,80	<b>76,70</b>	72,70	76,50	74,28
Permissions_only	70,90	70,50	73,80	71,70	<b>74,80</b>	72,34
Provider_only	62,80	55,30	61,30	62,20	<b>63,10</b>	60,94
Services_only	72,40	70,90	72,60	70,00	<b>73,30</b>	71,84
Sint_only	61,50	54,60	59,30	<b>62,20</b>	60,10	59,54
Receivers_only	71,50	67,30	<b>72,40</b>	67,90	71,30	70,08
Rint_only	73,50	67,20	69,30	66,90	<b>73,90</b>	70,16
Provider_Permissions_Intent	72,20	71,80	75,50	71,50	<b>76,30</b>	73,46
Permissions_Intent	71,50	71,40	75,00	71,30	<b>75,70</b>	72,98
Provider_Permissions	72,20	71,70	74,40	72,80	<b>75,40</b>	73,30
Provider_Intent	81,90	79,80	83,00	79,20	<b>84,10</b>	81,60
Intent_only	82,80	80,40	82,40	79,60	<b>84,20</b>	81,88
<b>MÉDIA</b>	72,16	69,48	72,98	70,67	74,06	71,87

VIRUS SHARE - Precisão %						
TIPO	ARV	KNN	SVM	BAYES	R F	MÉDIA
<b>TODAS FEATURES</b>	87,90	87,80	90,50	80,40	<b>91,60</b>	87,64
Permissions_only	85,80	85,90	90,00	80,10	<b>90,40</b>	86,44
Provider_only	81,50	68,20	81,10	74,90	<b>81,80</b>	77,50
Services_only	81,40	<b>83,10</b>	82,30	78,40	83,00	81,64
Sint_only	86,50	83,80	86,30	75,30	<b>88,40</b>	84,06
Receivers_only	82,90	81,90	83,40	78,80	<b>84,10</b>	82,22
Rint_only	87,20	82,00	<b>88,00</b>	83,10	<b>88,00</b>	85,66
Provider_Permissions_Intent	86,40	87,70	<b>90,50</b>	80,20	81,20	85,20
Permissions_Intent	86,60	87,50	90,50	80,30	<b>91,40</b>	87,26
Provider_Permissions	86,80	87,00	90,10	79,70	<b>91,20</b>	86,96
Provider_Intent	86,10	83,00	86,90	76,80	<b>89,00</b>	84,36
Intent_only	86,40	84,20	87,60	76,50	<b>89,50</b>	84,84
<b>MÉDIA</b>	85,46	83,51	87,27	78,71	87,47	84,48

GITHUB - Precisão %						
TIPO	ARV	KNN	SVM	BAYES	R F	MÉDIA
TODAS FEATURES	85,00	85,00	84,80	87,40	<b>88,00</b>	86,04
Permissions_only	85,60	85,50	86,40	83,10	<b>89,90</b>	86,10
Provider_only	75,60	67,20	73,70	73,00	<b>77,60</b>	73,42
Services_only	75,70	73,50	75,40	73,80	<b>77,00</b>	75,08
Sint_only	75,70	69,50	76,60	<b>77,70</b>	<b>77,70</b>	75,44
Receivers_only	82,30	77,70	80,80	79,50	<b>82,60</b>	80,58
Rint_only	85,00	80,90	82,50	85,60	<b>87,80</b>	84,36
Provider_Permissions_Intent	82,00	84,40	84,80	85,60	<b>88,00</b>	84,96
Permissions_Intent	83,40	84,40	84,60	84,80	<b>87,80</b>	85,00
Provider_Permissions	85,00	86,80	85,60	82,80	<b>88,40</b>	85,72
Provider_Intent	83,80	77,40	80,40	81,20	<b>85,40</b>	81,64
Intent_only	84,80	81,80	85,00	84,20	<b>85,80</b>	84,32
MÉDIA	81,99	79,51	81,72	81,56	84,67	81,89

ALL - Precisão %						
TIPO	ARV	KNN	SVM	BAYES	R F	MÉDIA
TODAS FEATURES	74,30	74,50	77,20	69,20	<b>78,60</b>	74,76
Permissions_only	74,40	73,00	75,40	67,70	<b>77,90</b>	73,68
Provider_only	77,10	60,70	<b>77,50</b>	71,30	77,40	72,80
Services_only	76,60	70,90	76,70	74,40	<b>77,80</b>	75,28
Sint_only	80,50	69,40	80,90	70,90	<b>81,70</b>	76,68
Receivers_only	77,90	73,80	77,40	73,40	<b>78,70</b>	76,24
Rint_only	75,60	70,00	<b>78,80</b>	75,20	78,70	75,66
Provider_Permissions_Intent	74,40	73,30	77,00	68,10	<b>79,10</b>	74,38
Permissions_Intent	74,30	72,20	76,50	68,00	<b>79,30</b>	74,06
Provider_Permissions	74,60	73,90	76,60	68,80	<b>78,20</b>	74,42
Provider_Intent	81,70	78,70	83,30	84,00	<b>84,10</b>	82,36
Intent_only	81,30	77,60	83,20	<b>83,70</b>	<b>83,70</b>	81,90
MÉDIA	76,89	72,33	78,38	72,89	79,60	76,02

Os critérios de avaliação para um detector de *Malware* é geralmente feita através de cálculo AUROC que é a área embaixo da curva ROC que tem relação com a sensibilidade e especificidade de um modelo. Outra forma de avaliação é através das métricas de uma matriz de confusão que são: verdadeiro positivo (TP/*True Positive*) e falso verdadeiro (TN/*True Negative*) que ocorrem quando o modelo acerta a previsão; e falso positivo (FP/*False Positive*) e falso negativo (FN/*False Negative*) que ocorrem quando o modelo erra a previsão [AZ13].

Quando se trata de detectores de *Malware* falsos positivos e falso negativos tem-se um peso diferente quando se analisa os detectores, pois esses dois tipos de erros trazem desfechos diferentes para o usuário. Um falso positivo em um detector de *Malware Android* bloqueia ou impede a instalação de um aplicativo benigno no dispositivo, impossibilitando o usuário de utilizá-lo. Já um falso negativo permite o funcionamento livre de um aplicativo malicioso dentro do dispositivo do usuário. Para avaliar os detectores construí uma tabela que possui os índices de falso positivo e falso negativo de cada modelo, as médias de cada classificador e as médias dos resultados.

**Tabela 3.6:** Tabelas que possuem o índice de falso positivo e falso negativo de cada modelo, as médias de cada classificador e as médias dos resultados. (Autor)

ALL - Falso Positivo e Falso Negativo %													
-	ARV		KNN		SVM		Bayes		R F		Media		
Tipos	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	
prov	8,62	7,69	9,54	10,15	8,62	7,69	34,15	0,31	8,62	7,38	13,91	6,64	
permission	12,58	14,36	10,04	14,36	11,31	12,33	28,08	3,43	10,8	11,94	14,56	11,28	
receiver	7,14	15,16	8,54	16,9	6,79	16,03	25,26	1,39	6,45	14,11	10,84	12,72	
RINT	13,6	10,25	17,31	11,48	14,31	9,54	24,03	4,06	13,6	8,83	16,57	8,83	
service	4,97	17,41	26,82	5,33	4,26	18,83	30,2	1,07	4,8	17,23	14,21	11,97	
SINT	28,25	10,41	10,41	13,38	11,15	11,52	32,71	1,86	12,27	9,67	18,96	9,37	
INT	11,65	5,22	10,26	7,83	6,61	7,3	7,13	6,26	10,78	2,96	9,29	5,91	
pro per int	12,72	13,22	8,82	15,74	7,81	14,36	29,22	2,02	9,82	11,08	13,68	11,28	
per pro	11,55	16,12	7,74	15,61	7,99	14,21	27,66	3,81	7,36	14,21	12,46	12,79	
pro int	10,18	6,68	8,35	10,02	7,85	9,02	7,85	7,51	9,02	5,34	8,65	7,71	
per int	12,47	13,22	9,19	15,24	8,31	13,85	29,22	2,02	10,2	11,84	13,88	11,23	
total	11,46	13,22	9,45	15,99	8,44	14,36	28,84	1,89	10,2	11,21	13,68	11,33	
MEDIA	12,1	11,91	11,37	12,67	8,62	12,42	25,36	2,97	9,49	10,48	13,39	10,09	

ANDROZOO - Falso Positivo e Falso Negativo %													
-	ARV		KNN		SVM		Bayes		R F		Media		
Tipos	FN	FP											
prov	11,9	28,57	2,38	30,95	11,9	30,95	9,52	33,33	9,52	32,14	9,04	31,19	
per	12,6	15,75	9,71	19,69	8,27	17,45	9,58	19,69	11,42	13,65	10,32	17,25	
receiver	7,92	19,25	7,17	23,02	6,42	20,38	4,15	26,79	6,04	19,62	6,34	21,81	
RINT	12,75	10,36	9,56	19,52	17,13	18,33	10,76	17,13	10,36	13,15	12,11	15,7	
service	5,08	26,78	5,76	26,78	3,73	26,78	5,42	24,07	5,08	25,08	5,01	25,9	
SINT	17,14	25,71	15,71	30	22,86	25,71	17,14	25,71	15,71	30	17,71	27,43	
INT	9,36	8,92	8,32	10,1	7,73	8,32	5,5	13,52	7,88	7,13	7,76	9,6	
pro per int	12,99	15,19	6,49	21,3	7,01	16,88	8,31	19,87	10,13	13,12	8,99	17,27	
per pro	12,73	15,32	6,36	21,43	7,14	16,88	8,57	19,87	10,39	12,86	9,04	17,27	
pro int	20,91	11,7	7,6	12,57	5,99	10,67	4,68	16,67	6,14	10,53	9,06	12,43	
per int	13,25	14,42	6,75	19,61	7,14	16,88	7,66	21,04	10,91	12,6	9,14	16,91	
total	12,73	14,81	5,71	22,86	6,62	15,71	5,58	21,69	8,96	13,25	7,92	17,66	
MEDIA	12,45	17,23	7,63	21,49	9,33	18,75	8,07	21,62	9,38	16,93	9,37	19,2	

VIRUSSHARE - Falso Positivo e Falso Negativo %													
-	ARV		KNN		SVM		Bayes		R F		Media		
Tipos	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	
prov	1,46	16,99	2,43	18,45	1,46	16,99	20,39	0,49	1,46	16,5	5,44	13,88	
permission	7,81	5,6	7,42	4,95	6,77	4,95	17,06	3,13	5,73	4,69	8,96	4,66	
receiver	5,05	14,98	5,7	15,47	4,23	14,98	17,43	0,49	4,89	13,84	7,46	11,95	
RINT	5,6	6,1	5,6	5,77	5,11	6,43	13,18	1,81	4,78	3,46	6,85	4,71	
service	13,14	0,96	3,04	16,19	2,4	16,19	30,45	0,64	12,18	1,44	12,24	7,08	
SINT	4,78	8,06	4,48	8,96	3,58	8,06	18,51	1,19	3,58	7,46	6,99	6,75	
INT	6,4	7,29	6,1	5,95	4,46	7,29	22,47	1,93	4,61	5,36	8,81	5,56	
pro per int	5,46	7,54	5,33	4,29	5,59	4,94	16,91	2,73	4,68	4,68	7,59	4,84	
per pro	6,89	6,76	5,85	4,42	5,46	4,68	15,6	2,86	4,94	4,16	7,75	4,58	
pro int	7,46	7,6	6,29	6,87	5,12	9,8	19,01	3,51	5,41	6,58	8,66	6,87	
per int	6,63	8,19	5,33	4,42	5,33	4,81	16,78	2,73	4,42	4,68	7,7	4,97	
total	5,98	6,89	5,72	4,42	4,81	5,07	17,17	2,08	3,9	4,81	7,52	4,65	
MEDIA	6,39	8,08	5,27	8,35	4,53	8,68	18,75	1,97	5,05	6,47	8	6,71	

GITHUB - Falso Positivo e Falso Negativo %													
-	ARV		KNN		SVM		Bayes		R F		Media		
Tipos	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	
prov	5,56	16,67	2,78	16,67	22,22	2,78	5,56	11,11	2,78	16,67	7,78	12,78	
permission	9,02	2,46	8,2	4,1	6,56	4,1	13,11	2,46	7,38	1,64	8,85	2,95	
receiver	1,22	17,07	6,1	19,51	1,22	13,41	2,44	15,85	1,22	12,2	2,44	15,61	
RINT	7,5	10	3,75	16,25	5	13,75	6,25	10	5	11,25	5,5	12,25	
service	1,27	26,58	20,25	2,53	1,27	26,58	1,27	27,85	0	24,05	4,81	21,52	
SINT	4	20	4	16	8	16	8	16	4	20	5,6	17,6	
INT	14,4	5,6	12	14,4	6,4	10,4	5,6	12,8	9,6	6,4	9,6	9,92	
pro per int	12,8	8,8	17,6	8,8	13,6	5,6	14,4	4,8	14,4	8	14,56	7,2	
per pro	10,4	7,2	12,8	4,8	10,4	5,6	16	1,6	11,2	4,8	12,16	4,8	
pro int	9,6	8	11,2	4,8	7,2	6,4	3,2	9,6	8	2,4	7,84	6,24	
per int	8	7,2	8,8	7,2	4,8	12	8,8	6,4	5,6	10,4	7,2	8,64	
total	12,8	9,6	16	8,8	12	6,4	8,8	6,4	12	5,6	12,32	7,36	
MEDIA	8,05	11,6	10,29	10,32	8,22	10,25	7,79	10,41	6,77	10,28	8,22	10,57	

### 3.4.2 ANÁLISE DE RESULTADOS

Para facilitar a análise das tabelas de precisão foram criadas duas tabelas auxiliares, tabelas 3.7 e 3.8 mostradas abaixo, com as médias do desempenho das *features* e algoritmos respectivamente. Na tabela 3.7 de *features* observa-se que os algoritmos que utilizaram “*Intents only*”, “*Providers Intents*” e “Todas as *features* disponíveis” tiveram um desempenho melhor em precisão. Já os que utilizaram “*Providers Only*” e “*Service Intents (Sint)*” tiveram os piores desempenhos. Com relação a tabela 3.8 pode-se observar que os dois algoritmos com melhor resultado foram *SVM* e *Random Forest* e os piores foram *Bayes* e *KNN*.

Mais uma observação interessante é que nos resultados da tabela 3.7 percebe-se que os valores diferenciam bastante em relação a mudança da base de dados, em que os detectores no

*AndroZoo* tiveram uma média de 71,87% de precisão e já no *VirusShare*, que á uma base parecida e com tamanho semelhante, a média foi de 84,48%.

**Tabela 3.7:** Tabela da Precisão Média de *Features* (Autor)

PRECISÃO MÉDIA DE FEATURES %					
TIPO	ANDROZOO	VIRUSHARE	GITHUB	ALL	MÉDIA TOTAL
TODAS FEATURES	74.28	87.64	86.04	74.76	80.68
Permissions_only	72.34	86.44	86.10	73.68	79.64
Provider_only	60.94	77.50	73.42	72.80	71.17
Services_only	71.84	81.64	75.08	75.28	75.96
Sint_only	59.54	84.06	75.44	76.68	73.93
Receivers_only	70.08	82.22	80.58	76.24	77.28
Rint_only	70.16	85.66	84.36	75.66	78.96
Provider_Permissions_Intent	73.46	85.20	84.96	74.38	79.50
Permissions_Intent	72.98	87.26	85.00	74.06	79.83
Provider_Permissions	73.30	86.96	85.72	74.42	80.10
Provider_Intent	81.60	84.36	81.64	82.36	82.49
Intents_only	81.88	84.84	84.32	81.90	83.24
MÉDIA	71.87	84.48	81.89	76.02	78.56

**Tabela 3.8:** Tabela da Precisão Média de *Algoritmos* (Autor)

PRECISÃO MÉDIA DE ALGORITMOS %					
ALGORÍTIMOS	ARV	KNN	SVM	BAYES	R F
ANDROZOO	72.16	69.48	72.98	70.67	74.06
VIRUSSHARE	85.46	83.51	87.27	78.71	87.47
GITHUB	81.99	79.51	81.72	81.56	84.67
ALL	76.89	72.33	78.38	72.89	79.60
MÉDIA	79.13	76.21	80.08	75.96	81.45

Para facilitar a visualização dos resultados de falso positivo e falso negativo foram criados, também, mais três tabelas, 3.9, 3.10 e 3.11, representadas abaixo. A primeira, 3.9 com relação a média dos valores de falso positivo das *features*; a segunda, 3.10 com relação a média dos valores de falso negativo das *features* e por último uma tabela 3.11, com as médias dos falsos positivos e falso negativos com relação aos algoritmos utilizados. Na tabela 3.9 observa-se que as *features* que apresentaram melhor desempenho foram: “*Permission Only*”, “*Intents Only*” e “*Provider Intens*” e as que apresentaram os piores desempenhos foram: “*Receiver Only*” e “*Services Only*”. Na segunda 3.10 observa-se que a “*Intents Only*”, “*Reciever Only*” e “*Provider Intents*” tiveram os melhores números e “*Sint*” e “*Permissions Only*” tem os piores desempenhos.

Já em se tratando de algorítimos, pode-se ver que na tabela 3.11, que os melhores algoritmos com relação a falso positivo foram “*Naive Bayes*” e “*Random Forest*” e os piores algorítimos foram *SVM* e *KNN*. Já com relação aos falsos negativos, os melhores foram *SVM* e “*Random Forest*” e os piores “*Naive Bayes*” e Árvore.

**Tabela 3.9:** Tabela de Média de FN em *Features* (Autor)

MEDIA DE FN EM FEATURES %					
BASES	ALL	ANDROZOO	GITHUB	VIRUSSHARE	MEDIA TOTAL
prov	13,91	9,04	7,78	5,44	9,0425
permission	14,56	10,32	8,85	8,96	10,6725
receiver	10,84	6,34	2,44	7,46	6,77
RINT	16,57	12,11	5,5	6,85	10,2575
service	14,21	5,01	4,81	12,24	9,0675
SINT	18,96	17,71	5,6	6,99	12,315
INT	9,29	7,76	9,6	8,81	8,865
pro per int	13,68	8,99	14,56	7,59	11,205
per pro	12,46	9,04	12,16	7,75	10,3525
pro int	8,65	9,06	7,84	8,66	8,5525
per int	13,88	9,14	7,2	7,7	9,48
total	13,68	7,92	12,32	7,52	10,36

**Tabela 3.10:** Tabela de Média de FP em *Features* (Autor)

MEDIA DE FP EM FEATURES %					
BASES	ALL	ANDROZOO	GITHUB	VIRUSSHARE	MEDIA TOTAL
prov	6,64	31,19	12,78	13,88	16,1225
permission	11,28	17,25	2,95	4,66	9,035
receiver	12,72	21,81	15,61	11,95	15,5225
RINT	8,83	15,7	12,25	4,71	10,3725
service	11,97	25,9	21,52	7,08	16,6175
SINT	9,37	27,43	17,6	6,75	15,2875
INT	5,91	9,6	9,92	5,56	7,7475
pro per int	11,28	17,27	7,2	4,84	10,1475
per pro	12,79	17,27	4,8	4,58	9,86
pro int	7,71	12,43	6,24	6,87	8,3125
per int	11,23	16,91	8,64	4,97	10,4375
total	11,33	17,66	7,36	4,65	10,25

**Tabela 3.11:** Tabela de Falso Positivo e Negativo entre Algoritmos (Autor)

FALSO POSITIVO E NEGATIVO ENTRE ALGORITMOS %										
Algoritmo	ARV		KNN		SVM		Bayes		R F	
	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP
XXX										
ALL	12,1	11,91	11,37	12,67	8,62	12,42	25,36	2,97	9,49	10,48
ANDROZOO	12,45	17,23	7,63	21,49	9,33	18,75	8,07	21,62	9,38	16,93
GITHUB	8,05	11,6	10,29	10,32	8,22	10,25	7,79	10,41	6,77	10,28
VIRUSSHARE	6,39	8,08	5,27	8,35	4,53	8,68	18,75	1,97	5,05	6,47
MEDIA TOTAL	9,74	12,20	8,64	13,20	7,67	12,52	14,99	9,24	7,67	11,04

## 4. CONCLUSÃO

Nesta monografia foi feito um estudo sobre a importância da seleção de *features* para detectores de *Malwares* em *Android*, escolhendo-se fazer detectores utilizando a metodologia baseada em permissões, que consiste na procura de maliciosidade através da análise estática do arquivo **Manifesto** do aplicativo. Para construção dos detectores foram utilizados cinco algoritmos de aprendizagem de máquina (Árvore Binária, KNN, SVM, *Naive Bayes* e *Random Forest*). Logo após, foi feita uma análise dos resultados obtidos que mostraram que o *Random Forest*, utilizando *features* relacionados com *Intents* e *Providers Intents* conseguem produzir bons resultados de precisão e baixos valores de falso positivo.

A análise também mostrou que outros algoritmos que utilizaram apenas as *features* permissões, como os detectores tradicionais antigos, apresentaram alta precisão durante os testes e também apresentaram baixo índice de falso positivo, porém em comparação com os detectores que utilizam *Intents* eles se mostraram menos eficientes.

Com esses resultados pode-se concluir, que a análise estática do arquivo manifesto consegue mostrar a maliciosidade de alguns arquivos, porém apenas com a extração de *features* do arquivo manifesto sem o refinamento do algoritmos não é suficiente para produzir um detector que consiga alcançar resultados estáveis acima de 95% e valores baixos de falsos positivo.

Em suma, quando deseja-se fazer um análise estática em aplicativos maliciosos *Android* utilizando detectores baseados em permissão recomenda-se analisar e utilizar “*Intents*” como parte das *features* nos detectores, pois eles conseguem provar a maliciosidades de aplicativos

### 4.1 TRABALHOS FUTUROS

O projeto desta monografia foi baseado em uma das várias metodologias de análise estática, como sugestão futura recomenda-se utilizar outras metodologias de análise de *Malware* em *Android* ou mesmo até combinar metodologias para conseguir diferentes resultados. Outra sugestão é utilizar bases alternativas de dados e comparar os resultados encontrados, refinar os algoritmos utilizados na monografia ou ainda utilizar algoritmos de seleção de *features*.

## REFERÊNCIAS

- [ABKLT16] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pages 468–471. IEEE, 2016
- [Anda] Arquitetura da plataforma. <https://developer.android.com/guide/platform?hl=pt-PT>. Accessed: 2019-11-24.
- [andb] Permissions overview. <https://developer.android.com/guide/topics/permissions/overview>. Accessed: 2019-11-24.
- [ASH+14] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck Drebin: Effective and explainable detection of android malware in your pocket. 02 2014.
- [ASKA16] Saba Arshad, Munam Shah, Abid Khan, and Mansoor Ahmed. Android malware detection protection: A survey. International Journal of Advanced Computer Science and Applications, 7, 02 2016.
- [ASKO11] Frank Ableson, Robi Sen, Chris King, and C Enrique Ortiz. Android in action. Manning Publications Co., 2011.
- [AVW+11] Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, Moutaz Alazab, and Ammar Alazab. Cybercrime: the case of obfuscated malware. In Global Security, Safety and Sustainability & e-Democracy, pages 204–211. Springer, 2011.
- [AZ13] Zarni Aung and Win Zaw. Permission-based android malware detection. International Journal of Scientific & Technology Research, 2(3):228–234, 2013.
- [BPK15] Mitul Bhatt, Hinaxi Patel, and Swati Kariya. A survey permission based mobile malware detection. International Journal of Computer Technology and Applications, 6:2, 10 2015.
- [Bur19] Andriy Burkov. The Hundred-Page Machine Learning Book. Andriy Burkov, 2019.
- [CF] Vinod Kumar Chandola and Huifen Fu. Market penetration strategy of smartphone companies from china for india market: A multiple-case study.
- [D+13] Anthony Desnos et al. Androguard-reverse engineering, malware and goodware analysis of android applications. URL code. google.com/p/androguard, 153, 2013.
- [DD16] Sumeet Dua and Xian Du. Data mining and machine learning in cybersecurity. Auerbach Publications, 2016.
- [DDTV+17] Anusha Damodaran, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H Austin, and Mark Stamp. A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques, 13(1):1–12, 2017.
- [Dig] Digital 2019: Global digital overview. <https://datareportal.com/reports/digital-2019-global-digital-overview>. Accessed: 2019-11-24.

- [fra] 10 organizations build 60 of russian toll fraud malware. <https://blog.lookout.com/10-organizations-build-60-of-russian-toll-fraud-malware>. Accessed: 2019-11-24.
- [GAF+15] André Ricardo Abed Grégio, Vitor Monte Afonso, Dario Simões Fernandes Fi- lho, Paulo Lício de Geus, and Mario Jino. Toward a taxonomy of malware beha- viors. *The Computer Journal*, 58(10):2758–2777, 2015.
- [HAC] Definição hacker. <https://www.significados.com.br/hacker/>. Accessed: 2019-11-24.
- [HAM19] Yasmin Salah Ibrahim Hamed, Sarah Nabil Abdullah AbdulKader, and Mostafa-Sami M Mostafa. Mobile malware detection: A survey. *International Journal of Computer Science and Information Security (IJCSIS)*, 17(1), 2019.
- [Har12] Peter Harrington. *Machine learning in action*. Manning Publications Co., 2012.
- [HYSA17] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1507–1515. ACM, 2017.
- [IDC] Smartphone market share. <https://www.idc.com/promo/mobile-market-share/os>. Accessed: 2019-11-24.
- [KJP19] Shreya Khemani, Darshil Jain, and Gaurav Prasad. Android malware detection techniques. In *Emerging Research in Computing, Information, Communication and Applications*, pages 449–457. Springer, 2019.
- [KRKP+16] Thomas Kluyver, Benjamin Ragan Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks. a publishing format for reproducible computational workflows, (<https://jupyter.org/>). 850:87–90, 2016.
- [LKG+17] Arash Habibi Lashkari, Andi Fitriah A Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 233–23309. IEEE, 2017.
- [LKS17] E Rutger Leukfeldt, Edward R Kleemans, and Wouter P Stol. Cybercriminal networks, social ties and online forums: social ties versus digital ties within phishing and malware networks. *The British Journal of Criminology*, 57(3):704–722, 2017.
- [LLH+16] Xiang Li, Jianyi Liu, Yanyu Huo, Ru Zhang, and Yuangang Yao. An android malware detection method based on androidmanifest file. In *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pages 239–243. IEEE, 2016.
- [LSY+18] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-an, and Heng Ye. Significant permission identification for machine-learning-based

- android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018
- [mal] Malwaresamples. <https://github.com/ashishb/android-malware>. Accessed: 2019-11-24.
- [Mil13] Nikola Miloševic'. History of malware. arXiv preprint arXiv:1302.5392, 2013.
- [mlr] Common malware types: Cybersecurity 101  
<https://www.veracode.com/blog/2012/10/common-malware-types-cybersecurity-101>. Accessed: 2019-11-24..
- [MRB19] Alex Matrosov, Eugene Rodionov, and Sergey Bratus. Rootkits and bootkits: reversing modern malware and next generation threats. No Starch Press, 2019.
- [Nel15] Fabio Nelli. Python data analytics. Berkeley: <https://pandas.pydata.org/> Apress, 2015
- [par] Definição parser. <https://educalingo.com/pt/dic-en/parser>. Accessed: 2019-11-24.
- [PYSW19] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In Proceedings of the Internet Measurement Conference, pages 478–485. ACM, 2019.
- [QNL+19] Junyang Qiu, Surya Nepal, Wei Luo, Lei Pan, Yonghang Tai, Jun Zhang, and Yang Xiang. Data-driven android malware intelligence: A survey. In International Conference on Machine Learning for Cyber Security, pages 183–202. Springer, 2019.
- [RFC13] Alessandro Reina, Aristide Fattori, and Lorenzo Cavallaro. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. EuroSec, April, 2013.
- [RS19] Gary Davis Raj Samani. McAfee mobile threat report. Technical report, 2019.
- [RTL09] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. Encyclopedia of database systems, pages 532–538, 2009. Script az. <https://github.com/ArtemKushnerov/az>. Accessed:
- [scr] Script az. <https://github.com/ArtemKushnerov/az>. Accessed: 2019-11-24.
- [SD14] Akanksha Sharma and Subrat Kumar Dash. Mining api calls and permissions for android malware detection. In International Conference on Cryptology and Network Security, pages 191–205. Springer, 2014
- [SS14] Ashu Sharma and Sanjay Kumar Sahay. Evolution and detection of polymorphic and metamorphic malwares: A survey. arXiv preprint arXiv:1406.7061, 2014.
- [SS18] Joshua Saxe and Hillary Sanders. Malware Data Science: Attack Detection and Attribution. No Starch Press, 2018.
- [Staa] Android's rise to smartphone dominance. <https://www.statista.com/chart/15561/smartphone-sales-by-os/>. Accessed: 2019-11-24.

- [STAb] The numbers behind google’s online empire. <https://www.statista.com/chart/9467/usage-of-google-services/>. Accessed: 2019- 11-22.
- [SWL12] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. Tracking concept drift in malware families. In Proceedings of the 5th ACM workshop on Security and artificial intelligence, pages 81–92. ACM, 2012.
- [Tch14] Franklin Tchakounté. Permission-based malware detection mechanisms on android: Analysis and perspectives. *Journal of Computer Science*, 1(2), 2014.
- [typ] Malware types and classifications. <https://www.lastline.com/blog/malware-types-and-classifications/>. Accessed: 2019-11-24.
- [VDA16] Wil Van Der Aalst. Data science in action. In *Process Mining*, pages 3–23. Springer, 2016.
- [vira] Virus share about. <https://virusshare.com/about.4n6>. Accessed: 2019-11-24.
- [virb] Virus total : How it works. <https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works>. Accessed: 2019- 11-24.
- [WGNF12] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Permission evolution in the android ecosystem. In Proceedings of the 28th Annual Computer Security Applications Conference, pages 31–40. ACM, 2012.
- [WWF+14] Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, and Xiangliang Zhang. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9(11):1869–1882, 2014
- [YS19] Majid Yar and Kevin F Steinmetz. *Cybercrime and society*. SAGE Publications Limited, 2019.
- [YY10] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In 2010 International conference on broadband, wireless computing, communication and applications, pages 297–300. IEEE, 2010
- [ZAYC17] Raima Zachariah, K Akash, Mohammed Sajmal Yousef, and Anu Mary Chacko. Android malware detection a survey. In 2017 IEEE international conference on circuits and systems (ICCS), pages 238–244. IEEE, 2017.
- [ZX19] Dali Zhu and Tong Xi. Permission-based feature scaling method for lightweight android malware detection. In International Conference on Knowledge Science, Engineering and Management, pages 714–725. Springer, 2019

## ANEXOS

### ANEXO 1 – Código ‘*Preprocessamento.ipynb*’

```
In [1]: import pandas as pd
import ast
```

```
In [2]: # Read Database.csv
data = pd.read_csv('ALL_Untreated.csv', sep = ',')
```

```
In [3]: data.head()
```

Out[3]:

	Numero	Malware	Name	PERMISSIONS	PROVIDER
0	2	1	D:\torrent\Malware\\VirusShare_002d0119f75a44...	[ANDROID.PERMISSION.READ_SYNC_SETTINGS, 'AND...	
1	3	1	D:\torrent\Malware\\VirusShare_003efc6a5bf24a...	[ANDROID.PERMISSION.NFC, 'ANDROID.PERMISSION...]	[COM.OPPO.MUSIC..PROVIDERS.MUSICPROVIDER] [CC...
2	4	1	D:\torrent\Malware\\VirusShare_003fbe2039c2b0...	[ANDROID.PERMISSION.READ_SYNC_SETTINGS, 'COM...	
3	5	1	D:\torrent\Malware\\VirusShare_003fd8b8a0cccd...	[ANDROID.PERMISSION.CHANGE_CONFIGURATION, 'A...	
4	6	1	D:\torrent\Malware\\VirusShare_004e490fd28272...	[ANDROID.PERMISSION.SET_WALLPAPER, 'ANDROID....	

```
In [4]: m = data.loc[data['Malware']==1]
m.shape
```

Out[4]: (3326, 9)

```
In [5]: b = data.loc[data['Malware']==0]
b.shape
```

Out[5]: (1588, 9)

```
In [6]: # Função Transformar Uma coluna com listas em um dataset
def Trans(col,df):
    y = ((df[col]).tolist())
    x = []
    for i in y:
        x.append(ast.literal_eval(i))
    df_col = pd.DataFrame(x)
    col_obj = df_col.stack()
    col_df = pd.get_dummies(col_obj)
    col_df = col_df.sum(level=0)

    return col_df
```

```
In [7]: ### Transformar Colunas em Dataframes
permissions_df = Trans('PERMISSIONS',data)
provider_df = Trans('PROVIDER',data)
services_df = Trans('SERVICES',data)
sint_df = Trans('SINT',data)
receivers_df = Trans('RECEIVERS',data)
rint_df = Trans('RINT',data)

In [8]: # Juntar Bases com Base antiga

# Todos os recursos mesma base
final = pd.concat([data, permissions_df, rint_df, receivers_df, sint_df, services_df, provider_df], axis=1)

# Base Com Permissions, Intents e Provider
pro_per_int_only = pd.concat([data,permissions_df,provider_df,rint_df,sint_df], axis=1)

# Base com Permissions, Intents
per_int_only = pd.concat([data,permissions_df,rint_df,sint_df], axis=1)

# Base com Permissions, Provider
per_pro_only = pd.concat([data,permissions_df,provider_df], axis=1)

# Base com Intents , Provider
pro_int_only = pd.concat([data,provider_df,rint_df,sint_df], axis=1)

# Base com Intents
int_only = pd.concat([data,rint_df,sint_df], axis=1)

# Bases Separadas
permissions_only = pd.concat([data,permissions_df], axis=1)
provider_only = pd.concat([data,provider_df], axis=1)
services_only = pd.concat([data,services_df], axis=1)
sint_only = pd.concat([data,sint_df], axis=1)
receivers_only = pd.concat([data,receivers_df], axis=1)
rint_only = pd.concat([data,rint_df], axis=1)

In [9]: # Droping colunas não necessarias
lista_dfs = [final,permissions_only,provider_only,services_only,sint_only,receivers_only,rint_only,pro_per_int_only,per_int_only,per_pro_only,pro_int_only,int_only]

for i in lista_dfs:
    i.drop(['PERMISSIONS','Numero','Name','PROVIDER','SERVICES','SINT', 'RECEIVERS','RINT'],axis = 1,inplace = True)

In [10]: # Remover Colunas nulas e rows com apenas 0
# Os dataframes do Alone são feitos para analisar os atributos entao removemos as linhas que nao tem
# Ja mixed temos dataframes hiposteses

mixed = [pro_per_int_only,per_int_only,per_pro_only,pro_int_only,int_only,final]

for i in mixed:
    i.fillna(0, inplace=True)

alone = [permissions_only,provider_only,services_only,sint_only,receivers_only,rint_only]

for i in lista_dfs:
    i.dropna(inplace=True)

In [11]: # Remover Linhas com apenas 0

final = final[(final.T != 0).any()]
permissions_only = permissions_only[(permissions_only.T != 0).any()]
provider_only = provider_only[(provider_only.T != 0).any()]
services_only = services_only[(services_only.T != 0).any()]
sint_only = sint_only[(sint_only.T != 0).any()]
receivers_only = receivers_only[(receivers_only.T != 0).any()]
rint_only = rint_only[(rint_only.T != 0).any()]

pro_per_int_only = pro_per_int_only[(pro_per_int_only.T != 0).any()]
per_int_only = per_int_only[(per_int_only.T != 0).any()]
per_pro_only = per_pro_only[(per_pro_only.T != 0).any()]
pro_int_only = pro_int_only[(pro_int_only.T != 0).any()]
int_only = int_only[(int_only.T != 0).any()]

In [12]: # SEE SHAPES
lista_dfs = [final,permissions_only,provider_only,services_only,sint_only,receivers_only,rint_only,pro_per_int_only,per_int_only,per_pro_only,pro_int_only,int_only]

for i in lista_dfs:
    print(i.shape)

(4914, 29347)
(4877, 2506)
(1388, 1432)
(3241, 9472)
(1525, 2136)
(3147, 8391)
(3089, 5415)
(4914, 11486)
(4914, 10055)
(4901, 3937)
(4523, 8981)
(4476, 7550)

In [13]: final.to_csv('AndroZoo_TREATED.csv')
pro_per_int_only.to_csv('pro_per_int_only.csv')
per_int_only.to_csv('per_int_only.csv')
per_pro_only.to_csv('per_pro_only.csv')
pro_int_only.to_csv('pro_int_only.csv')
int_only.to_csv('int_only.csv')
permissions_only.to_csv('permissions_only.csv')
provider_only.to_csv('provider_only.csv')
services_only.to_csv('services_only.csv')
sint_only.to_csv('sint_only.csv')
receivers_only.to_csv('receivers_only.csv')
rint_only.to_csv('rint_only.csv')
```

## ANEXO 2 – Código ‘AI.ipynb’

```
In [1]: import pandas as pd
In [2]: # Adicionar Base CSV Pre processada desejada
df = pd.read_csv('provider_only.csv')
In [3]: df.head()
Out[3]:
   Unnamed: 0 Malware ABBI.IO.ABBISDK.PROVIDERS.MULTIPROCESSPERFENCESPROVIDER ANDRO.PAKISTANTV.BD_PROVIDER ANDROID.PPMEDIA.PROVI
0           1      1                         0.0                  0.0
1           9      1                         0.0                  0.0
2          10      1                         0.0                  0.0
3          15      1                         0.0                  0.0
4          18      1                         0.0                  0.0
5 rows × 1433 columns
In [4]: # Gerar Dataframe com Nomes e Vezes que o valor repetiu
x = df.sum()
l1 = x.tolist()
l2 = df.columns.tolist()
fd = pd.DataFrame(list(zip(l2, l1)),columns =['Name', 'val'])
In [5]: fd
Out[5]:
   Name      val
0      Unnamed: 0  3026270.0
1      Malware     650.0
2  ABBI.IO.ABBISDK.PROVIDERS.MULTIPROCESSPERFERENCE...     1.0
3      ANDRO.PAKISTANTV.BD_PROVIDER     1.0
4  ANDROID.PPMEDIA.PROVIDER.MEDIAPROVIDER     4.0
...
1428    TV.DANMAKU.BILI.PROVIDERS.BILISTATUSPROVIDER     1.0
1429      VN.MECORPIWIN.MYFILECONTENTPROVIDER     1.0
1430      WIKEM.CHRIS.WIKEMV3.DICTIONARYPROVIDER     1.0
1431      WSJ.UI.SEARCH.SEARCHSUGGESTIONPROVIDER     1.0
1432      YOU.IN.SPARK.ENERGY.EBPROVIDER     1.0
1433 rows × 2 columns
In [6]: fd.shape
Out[6]: (1433, 2)
In [7]: # Numero de recursos chamados mais de uma vez (MAIS DE 12000 recursos não são chamados)
umavez = fd.loc[fd['val'] > 1]
print("INFO Dataset: ", umavez.shape)
INFO Dataset: (350, 2)
In [8]: # Numero de recursos com ANDROID.PERMISSION no Nome
android = fd[(fd['Name'].str.contains("ANDROID.PRO"))]
print("INFO Dataset: ", android.shape)
INFO Dataset: (15, 2)
In [9]: # Numero de recursos com ANDROID.PERMISSION no Nome
androidp = fd[(fd['Name'].str.contains("ANDROID.PER"))]
print("INFO Dataset: ", androidp.shape)
INFO Dataset: (1, 2)
In [10]: # Numero de recursos com COM.ANDROID no Nome
com = fd[(fd['Name'].str.contains("COM.ANDROID"))]
print("INFO Dataset: ", com.shape)
INFO Dataset: (14, 2)
In [11]: # Numero de recursos com GOOGLE.ANDROID no Nome
google = fd[(fd['Name'].str.contains("GOOGLE.ANDROID"))]
print("INFO Dataset: ", google.shape)
INFO Dataset: (33, 2)
In [12]: # Unir condições
new = pd.concat([umavez, android, google, com, androidp])
new = new.drop_duplicates()
print("INFO Dataset: ", new.shape)
INFO Dataset: (409, 2)
In [13]: # LISTA DE PERMISSOES PARA O POC MELHORADO (ANDROID E >2)
important = new['Name'].tolist()
In [14]: jumbo = df[important]
print("INFO Dataset: ", jumbo.shape)
INFO Dataset: (1388, 409)
```

```
In [15]: # Remover possiveis linhas com apenas zeros
jumbo = jumbo[(jumbo.T != 0).any()]

In [16]: # Shuffle and UnderSampling (Redimensionar Base)
shuffled_df = jumbo.sample(frac=1,random_state=42)
mal_df = shuffled_df.loc[shuffled_df['Malware'] == 1]
benign_df = shuffled_df.loc[shuffled_df['Malware'] == 0]

if mal_df.shape[0] < benign_df.shape[0]:
    benign_df = shuffled_df.loc[shuffled_df['Malware'] == 0].sample(n=mal_df.shape[0],random_state=42)
else:
    mal_df = shuffled_df.loc[shuffled_df['Malware'] == 1].sample(n=benign_df.shape[0],random_state=42)

In [17]: # CHECK IF MAL = BENI
mal_df.shape == benign_df.shape

Out[17]: True

In [18]: norma = pd.concat([mal_df, benign_df])

In [19]: norma.shape

Out[19]: (1300, 409)

In [ ]: #####
In [20]: # TECNICAS DE APRENDIZAM DE MAQUINA

In [21]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

In [22]: # SEPARAR VALORES RECURSOS e VALOR da CLASSE
x = norma.values[:,2:norma.shape[1]]
y = norma.values[:,1]
y = y.astype('int')

In [23]: # Example Train Test Split (70-30)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state= 100)

In [24]: # Meu Classificador
def Classificador(tipo,x_train,y_train,x_test,y_test):
    if tipo == 'arv':
        clf = DecisionTreeClassifier()
    elif tipo == 'knn':
        clf = KNeighborsClassifier(n_neighbors=5)
    elif tipo == 'svm':
        clf = SVC(kernel= 'linear',gamma = 'auto')
    elif tipo == 'bb':
        clf = BernoulliNB()
    elif tipo == 'rf':
        clf = RandomForestClassifier(n_jobs=20, random_state=0 , n_estimators = 100)
    else:
        print ('WHAT ?')

    clf.fit(x_train,y_train)
    predict = clf.predict(x_test)
    score = accuracy_score(y_test,predict)*100

    return (predict,score)

In [25]: # Simple Arvore
predict_arv, score = Classificador('arv',x_train,y_train,x_test,y_test)
print('Precisão é ',score)

Precisão é  83.6923076923077

In [26]: # Simple KNN
predict_knn, score = Classificador('knn',x_train,y_train,x_test,y_test)
print('Precisão é ',score)

Precisão é  80.3076923076923

In [27]: # Simple SVM
predict_svm, score = Classificador('svm',x_train,y_train,x_test,y_test)
print('Precisão é ',score)

Precisão é  83.6923076923077

In [28]: # Simple Bayes
predict_bb, score = Classificador('bb',x_train,y_train,x_test,y_test)
print('Precisão é ',score)

Precisão é  65.53846153846153

In [29]: # Simples RANDOM FOREST
predict_rf, score = Classificador('rf',x_train,y_train,x_test,y_test)
print('Precisão é ',score)
```

