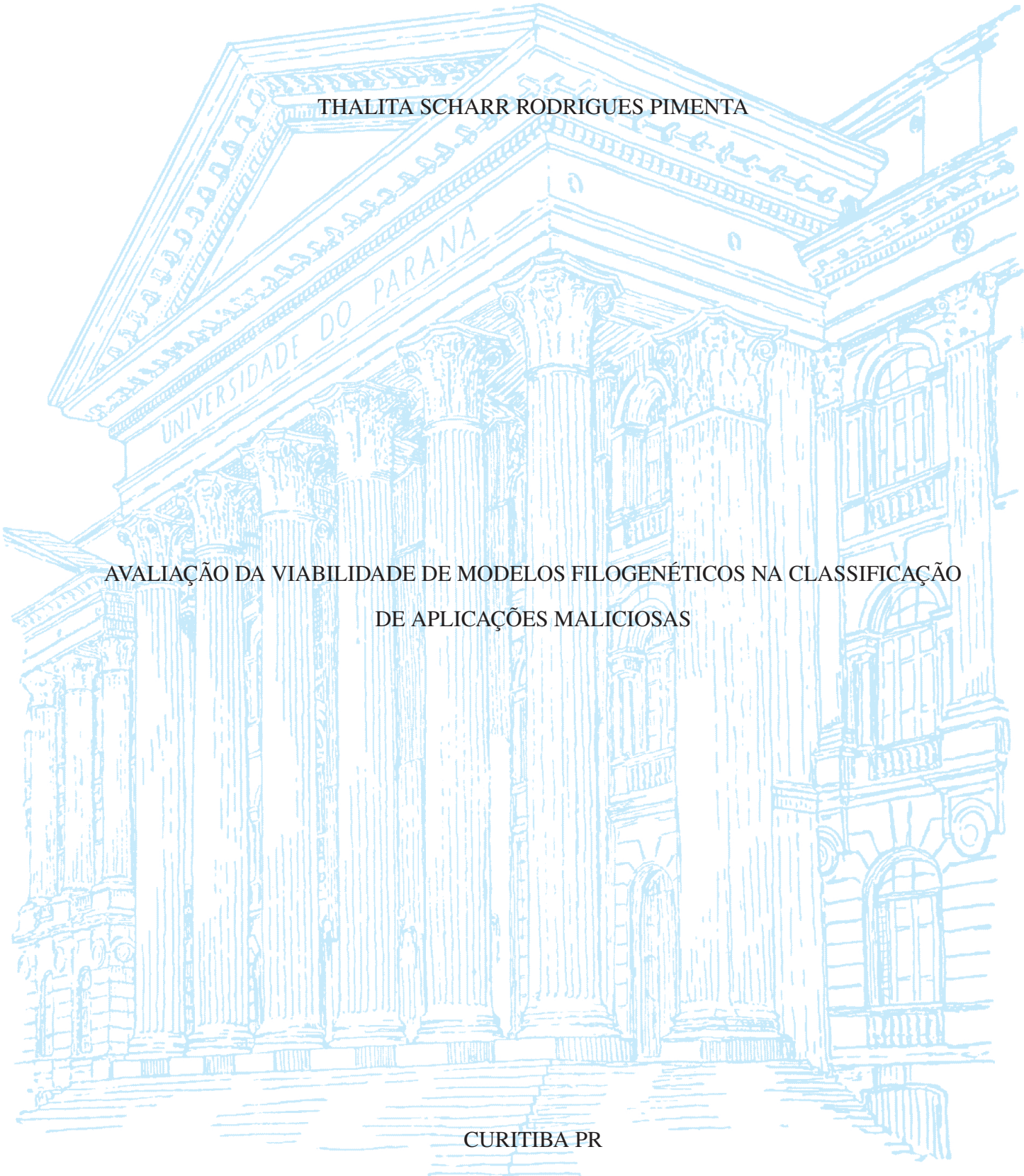
UNIVERSIDADE FEDERAL DO PARANÁ

THALITA SCHARR RODRIGUES PIMENTA



AVALIAÇÃO DA VIABILIDADE DE MODELOS FILOGENÉTICOS NA CLASSIFICAÇÃO DE APLICAÇÕES MALICIOSAS

CURITIBA PR

2023

THALITA SCHARR RODRIGUES PIMENTA

AVALIAÇÃO DA VIABILIDADE DE MODELOS FILOGENÉTICOS NA CLASSIFICAÇÃO DE APLICAÇÕES MALICIOSAS

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: André Ricardo Abed Grégio.

CURITIBA PR

2023

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Pimenta, Thalita Scharr Rodrigues
Avaliação da viabilidade de modelos filogenéticos na classificação de aplicações maliciosas / Thalita Scharr Rodrigues Pimenta. – Curitiba, 2023.
1 recurso on-line : PDF.

Tese (Doutorado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.

Orientador: André Ricardo Abed Grégio

1. Malware. 2. Filogenia. 3. Métricas de software. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Informática. III. Grégio, André Ricardo Abed. IV . Título.

Bibliotecário: Leticia Priscila Azevedo de Sousa CRB-9/2029

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **THALITA SCHARR RODRIGUES PIMENTA** intitulada: **AVALIAÇÃO DA VIABILIDADE DE MODELOS FILOGENÉTICOS NA CLASSIFICAÇÃO DE APLICAÇÕES MALICIOSAS**, sob orientação do Prof. Dr. ANDRÉ RICARDO ABED GRÉGIO, que após terem inquirido a aluna e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutora está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 03 de Fevereiro de 2023.

Assinatura Eletrônica
17/05/2023 12:08:35.0
ANDRÉ RICARDO ABED GRÉGIO
Presidente da Banca Examinadora

Assinatura Eletrônica
17/05/2023 10:16:52.0
ROBSON DE OLIVEIRA ALBUQUERQUE
Avaliador Externo (UNIVERSIDADE DE BRASÍLIA)

Assinatura Eletrônica
16/05/2023 11:25:05.0
NATASHA MALVEIRA COSTA VALENTIM
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
16/05/2023 11:52:45.0
LUIZ EDUARDO SOARES DE OLIVEIRA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
16/05/2023 11:14:36.0
KATIA ROMERO FELIZARDO SCANNAVINO
Avaliador Externo (UNIVERSIDADE TECNOLóGICA FEDERAL DO PARANá - UTFPR)

Para Adilson, Lara e Luna.

AGRADECIMENTOS

Desejo exprimir os meus agradecimentos a todos aqueles que, de alguma forma, permitiram que esta tese se concretizasse.

Em primeiro lugar quero agradecer imensamente ao Prof. Dr. André Grégio, pela oportunidade, pelas orientações e especialmente pela paciência nessa longa jornada.

Aos meus colegas de laboratório, especialmente Marcus, Fabrício e Tamy, muito obrigada ! Agradeço o bom convívio, as boas discussões e, a alegria que por vezes se instalava.

Ao Instituto Federal do Paraná, pela licença para pós-graduação concedida e por todo o apoio.

E, finalmente agradeço, à minha família: - Aos meus pais, Vera e Valdemir, e à minha irmã Aline. Muito do que faço e sou, não seria ou faria, sem o apoio e carinho de vocês.

- Ao Adilson, meu marido, com amor, pelo permanente incentivo e preocupação com que sempre acompanhou este meu trabalho. Agradeço ainda a paciência e amor demonstrados nos meus momentos menos bons.

- Às minhas filhas, Lara e Luna, nascidas na reta final do doutorado, de quem retirei muita atenção, paciência e acompanhamento. Para vocês, meus amores, o meu eterno obrigado.

RESUMO

Milhares de códigos maliciosos são criados, modificados com apoio de ferramentas de automação e liberados diariamente na rede mundial de computadores. Entre essas ameaças, *malware* são programas projetados especificamente para interromper, danificar ou obter acesso não autorizado a um sistema ou dispositivo. Para facilitar a identificação e a categorização de comportamentos comuns, estruturas e outras características de *malware*, possibilitando o desenvolvimento de soluções de defesa, existem estratégias de análise que classificam *malware* em grupos conhecidos como famílias. Uma dessas estratégias é a Filogenia, técnica baseada na Biologia, que investiga o relacionamento histórico e evolutivo de uma espécie ou outro grupo de elementos. Além disso, a utilização de técnicas de agrupamento em conjuntos semelhantes facilita tarefas de engenharia reversa para análise de variantes desconhecidas. Uma variante se refere a uma nova versão de um código malicioso que é criada a partir de modificações de *malware* existentes. O presente trabalho investiga a viabilidade do uso de filogenias e de métodos de agrupamento na classificação de variantes de *malware* para plataforma Android. Inicialmente foram analisados 82 trabalhos correlatos para verificação de configurações de experimentos do estado da arte. Após esse estudo, foram realizados quatro experimentos para avaliar uso de métricas de similaridade e de algoritmos de agrupamento na classificação de variantes e na análise de similaridade entre famílias. Propôs-se então um Fluxo de Atividades para Agrupamento de *malware* com o objetivo de auxiliar na definição de parâmetros para técnicas de agrupamentos, incluindo métricas de similaridade, tipo de algoritmo de agrupamento a ser utilizado e seleção de características. Como prova de conceito , foi proposto o *framework* Androidgyny para análise de amostras, extração de características e classificação de variantes com base em medóides (elementos representativos médios de cada grupo) e características exclusivas de famílias conhecidas. Para validar o Androidgyny foram feitos dois experimentos: um comparativo com a ferramenta correlata Gefdroid e outro, com exemplares das 25 famílias mais populosas do *dataset* Androzoo.

Palavras-chave: Análise de *malware*, classificação de variantes de *malware*, similaridade

ABSTRACT

Thousands of malicious codes are created, modified with the support of tools of automation and released daily on the world wide web. Among these threats, malware are programs specifically designed to interrupt, damage, or gain access unauthorized access to a system or device. To facilitate identification and categorization of common behaviors, structures and other characteristics of malware, enabling the development of defense solutions, there are analysis strategies that classify malware into groups known as families. One of these strategies is Phylogeny, a technique based on the Biology, which investigates the historical and evolutionary relationship of a species or other group of elements. In addition, the use of clustering techniques on similar sets facilitates reverse engineering tasks for analysis of unknown variants. a variant refers to a new version of malicious code that is created from modifications of existing malware. The present work investigates the feasibility of using phylogenies and methods of grouping in the classification of malware variants for the Android platform. Initially 82 related works were analyzed to verify experiment configurations of the state of the art. After this study, four experiments were carried out to evaluate the use of similarity measures and clustering algorithms in the classification of variants and in the similarity analysis between families. In addition to these experiments, a Flow of Activities for Malware grouping with five distinct phases. This flow has purpose of helping to define parameters for clustering techniques, including measures of similarity, type of clustering algorithm to be used and feature selection. After defining the flow of activities, the Androidgyny framework was proposed, a prototype for sample analysis, feature extraction and classification of variants based on medoids and unique features of known families. To validate Androidgyny were Two experiments were carried out: a comparison with the related tool Gefdroid and another with copies of the 25 most populous families in the Androzoo dataset.

Keywords: Malware analysis, malware variant classification, similarity

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Programas maliciosos (*malware*) são criados a uma taxa de milhares de espécimes por dia devido a ferramentas de criação automática e à adaptação de códigos maliciosos conhecidos. Existe uma urgência de métodos efetivos para categorização automática de softwares entre maliciosos e não-maliciosos. Na investigação e análise de *malware*, o termo variante refere-se à atualizações baseadas em códigos maliciosos conhecidos. A determinação dó rótulo de um arquivo malicioso, geralmente, é apresentada como uma estrutura hierárquica, que consiste do tipo, plataforma, família e abreviação/numeração da variante da família (Chakraborty et al., 2017a). Família de *malware* é um grupo de elementos que compartilham características e comportamentos em comum, sendo utilizada na classificação dessas ameaças e no desenvolvimento de soluções anti-malware.

Um dos problemas apontados na análise de *malware* relaciona-se aos nomes das variantes. Uma prática comum é criar um nome para cada amostra de *malware* diferente que é analisada. Como a nomeação é um processo subjetivo, a maioria das empresas de software antivírus desenvolvem estratégias diferentes para dar nomes às suas amostras, o que resulta em muitos exemplares de *malware* iguais sendo referenciados com nomes distintos (Caldas, 2016). Houve tentativas de padrões e esquemas, mas na prática, as dificuldades nas análises e classificações ainda continuam (Bontchev, 2004).

Além da dificuldade de padrão de rotulação, outro problema são os resultados genéricos, os quais são ineficazes para tanto para tomada de decisão e mitigação da ameaça encontrada quanto para técnicas de aprendizado de máquina (Rashed e Suarez-Tangil, 2021). Na Figura 1.1 pode ser visualizado o resultado de uma consulta de uma APK maliciosa, com destaques para rótulos genéricos.

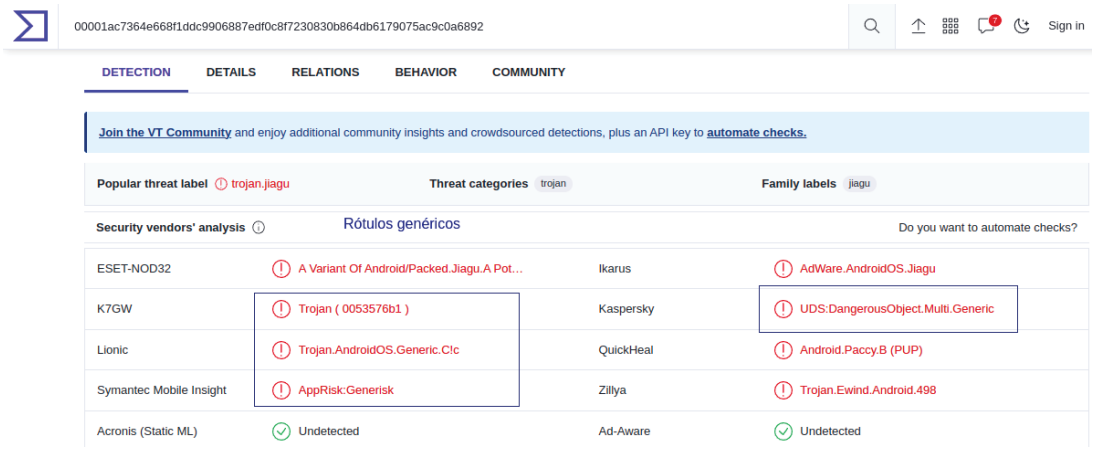


Figura 1.1: Tela de resultado da pesquisa de *hash* de *apk* no Vírus Total. Fonte: AVTest (2022)

Desse modo, além de saber quais comportamentos um *malware* apresenta, como roubar informações bancárias do usuário ou fazer *download* de pacotes prejudiciais, outros tipos de classificações são realizadas.

A classificação de *malware* não é uma pesquisa recente, datando do começo da década de 90. Grande parte dos trabalhos publicados nessa área tem como objetivo principal separar amostras maliciosas das não-maliciosas ou classificar amostras de código malicioso em famílias. Uma

família de *malware* contém espécimes (variantes) que implementam as mesmas funcionalidades ou que possuem comportamentos ou estruturas similares.

Classificar amostras de *malware* em famílias ajuda na defesa, remoção e análise de *malware*. Por exemplo, *malware* similares podem ter pontos fracos, estruturas e efeitos semelhantes, todos os quais podem ser identificados mediante melhores métodos de detecção (Seideman et al., 2014).

Uma forma de estudo de famílias e de seus relacionamentos e hierarquias é a análise filogenética. A filogenia, assim como na Biologia, é o relacionamento evolucionário entre um grupo de espécies. Em uma árvore filogenética, grupos possuem exemplares que compartilham um ancestral comum (Ragonnet-Cronin et al., 2013). Em *clustering* ou agrupamentos, uma das técnicas baseadas em distância utilizadas dentro da filogenia, apesar de haver similaridade entre os indivíduos, essa relação de parentesco não é obrigatória. Outras técnicas de construções filogenéticas incluem métodos baseados em parcimônia, máxima verossimilhança e abordagens bayesianas (Nasıbov e Kandemır-Cavas, 2011).

Além disso, a maioria dos problemas existentes em identificação, classificação e nomeação de programas maliciosos pode ser melhorada utilizando um sistema de modelagem filogenética de *malware* (Canfora et al., 2016). Essa melhoria se deve ao fato que a maioria dos novos *malware* são derivados ou compostos por outros *malware* já conhecidos. Por meio da detecção de funcionalidades, estruturas e pontos fracos semelhantes, desenvolver uma "vacina"se torna menos complexo e demorado. Portanto, essas informações permitem respostas mais imediatas e também permitem aos pesquisadores entender as novas variantes mais rapidamente.

Assim, dentro da classificação em famílias, no estudo da evolução de *malware* são abordadas quatro diferentes classes de problemas (Canfora et al., 2016):

1. **Análise filogenética**: visa reconstruir árvores de indivíduos considerando relações de ancestralidade entre espécies;

2. **Reconstrução da linhagem**: provê um modelo para reconstruções históricas de parentescos entre indivíduos;

3. **Identificação de variantes**: permite agrupar códigos maliciosos desconhecidos em famílias conhecidas ou novas famílias, otimizando a criação de "vacinas"; e

4. **Pesquisa de clonagem de código**: possibilita detecção de similaridade entre códigos, auxiliando na identificação de mutações, reuso de funções maliciosas e consequentemente no agrupamento de exemplares com comportamentos similares.

Em relação às classes de problemas supramencionadas, devido à relevância da filogenia, a presente tese foca na análise filogenética e na identificação de variantes de programas maliciosos. Por ser pertinente à filogenia e às famílias, a identificação de variantes foi selecionada. Partindo desses pressupostos, este trabalho investiga a viabilidade do uso de filogenias e de métodos de agrupamento na classificação de variantes de *malware* para plataforma Android.

1.1 MOTIVAÇÃO E QUESTÕES NORTEADORAS

Por muitos anos, a categorização em famílias de *malware* tem sido feita principalmente por analistas humanos, mediante um processo no qual memorização, estudo de descrição de bibliotecas e pesquisa de coleções de amostras são tipicamente requeridos. Esse processo manual consome tempo e é exaustivo (Ye et al., 2010b).

Em um cenário de mundo real, há influxo ininterrupto de amostras de códigos maliciosos recebidas. Em relação ao número de variantes de famílias de *malware* Windows, a quantidade de amostras identificadas variou entre 20 a 120 milhões no período de 2013-2023 (AV-Test, 2023). Porém, o uso desse sistema operacional tem diminuído com a ascensão da plataforma Android. Em 2013, o sistema operacional Windows tinha mais de 74% do mercado. Com o crescimento e a consolidação do Android, os números de uso do sistema da Microsoft hoje se encontram em menos de 30%. Informações sobre a percentagem do mercado de 2013 a 2023 podem ser visualizadas na Figura 1.2.

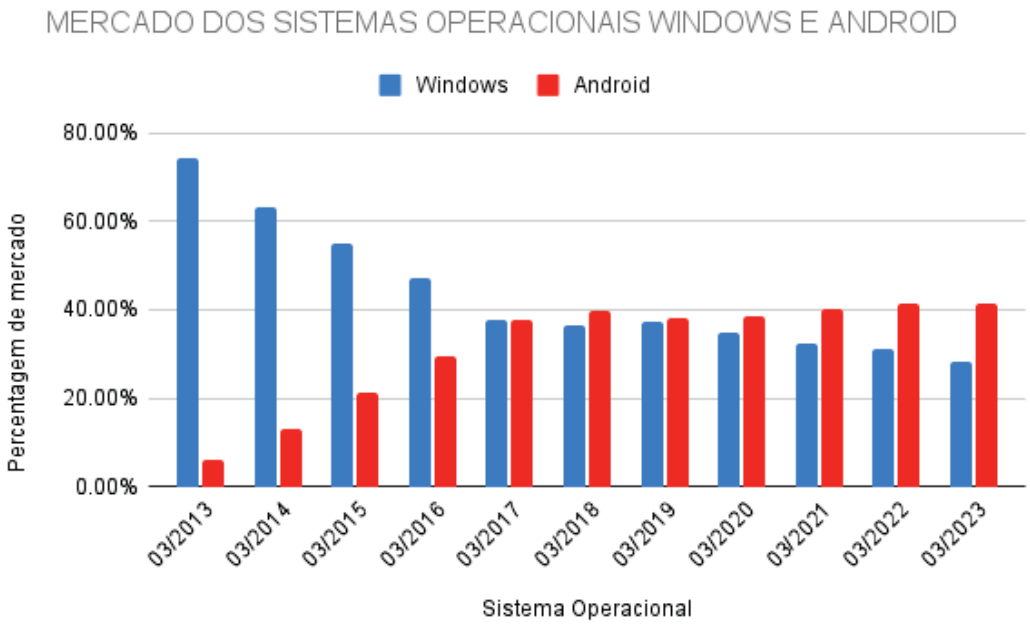


Figura 1.2: Variação da percentagem de predominância do Windows e Android no mercado de 2013-2023. Fonte: Statcounter (2023)

Apesar da quantidade de *malware* para Android ser menor, o número de dispositivos e de usuários dessa plataforma são predominantes no mercado. Conforme pode ser visto na Figura 1.3, o número de novas variantes de *malware* focadas no sistema operacional Android já passou de seis milhões, em 2016 e 2017. Nos últimos anos, apesar da quantidade diminuir ainda passa de três milhões de variantes liberadas nos mercados oficiais e paralelos de aplicativos. Devido ao crescimento do número de dispositivos e de aplicações maliciosas, a presente pesquisa tem como foco variantes de *malware* relacionados à plataforma Android.

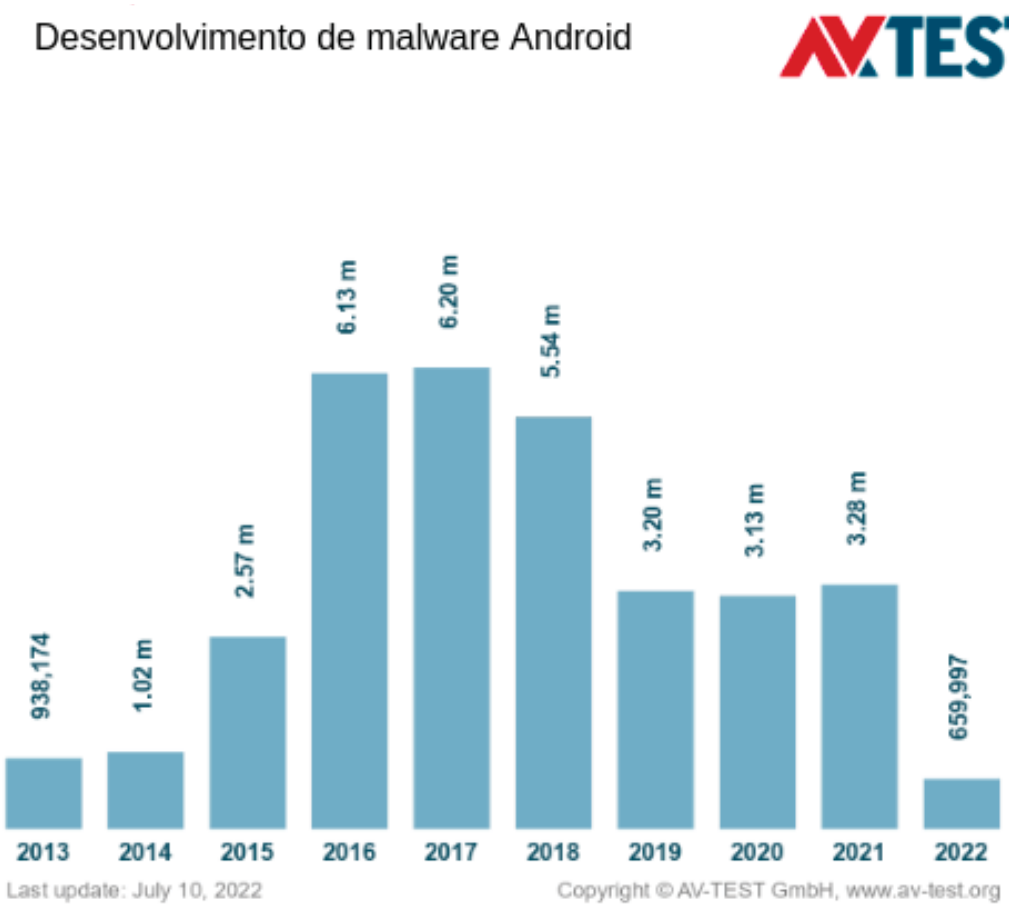


Figura 1.3: Quantidade de variantes de códigos maliciosos focados em Android descobertas nos últimos anos. Fonte: Adaptado de AVTest (2022)

Devido a grande quantidade de novos códigos maliciosos que são liberados diariamente, na literatura de análise de *malware*, a técnica de *clustering* ou agrupamento é amplamente utilizada (Rashed e Suarez-Tangil, 2021). Entretanto, devido a diversos parâmetros necessários aos algoritmos, tais como número de *clusters*, matriz de atributos ou matriz de similaridade/dissimilaridade, inicialmente as seguintes complexidades são encontradas:

- A escolha do algoritmo de agrupamento que melhor se adapta ao conjunto de exemplares e que não generalize os modelos aos *datasets* utilizados
- A medida utilizada para calcular a similaridade/dissimilaridade entre os elementos do conjunto pode impactar no resultado do agrupamento a partir dos atributos da amostra que foram utilizados
- Quando utilizados em agrupamentos exemplares não-maliciosos (*goodware*), em conjunto com *malware*, serão classificados em *clusters* separados ou com famílias de *malware*?

Visando contribuir à pesquisa da área com pontos mencionados e para o desenvolvimento do tema da pesquisa, foram estabelecidas as seguintes Questões norteadoras:

- **QP1**- Como as métricas influem na separação das famílias?
- **QP2**- Por que o agrupamento falha para determinados rótulos/famílias?

Na próxima seção são apresentados os objetivos desta tese.

1.2 OBJETIVOS

O objetivo geral da presente tese é propor um *framework* para classificação de variantes de *malware* Android em famíliasPara alcançar tal objetivo, os seguintes objetivos específicos foram definidos:

- Investigar as técnicas de agrupamento e seu funcionamento para descobrir limitações quando da aplicação em análise de códigos maliciosos
- Compreender o modo de operação e os resultados alcançados pelo estado da arte em filogenia e agrupamento de *malware* para classificação de famílias
- Propor e explicar um fluxo de atividades para agrupamento de variantes de *malware*
- Propor um *framework* de agrupamento de APKs maliciosos
- Avaliar o *framework* proposto e comparar a sua eficácia com o estado da arte da literatura

1.3 ORGANIZAÇÃO DA TESE

O restante deste documento está organizado da seguinte forma: no Capítulo 2 são apresentadas informações sobre a plataforma Android, informações sobre técnicas de agrupamento e filogenia aplicados à classificação de *malware*. No Capítulo 3 é apresentado o estado-da-arte e são analisados 82 trabalhos correlatos, investigando informações que incluem métodos de agrupamento, medidas de similaridade e número de amostras estudadas. Os experimentos iniciais de análise de medidas de similaridade e de métodos de agrupamento e o fluxo de atividades desenvolvido são apresentados no Capítulo 4. O *framework* Androidgyny é demonstrado no Capítulo 5 e o Capítulo 6 aborda os experimentos realizados para sua validação. Por fim, no Capítulo 7 é apresentada a Discussão e no Capítulo 8 apresentam-se as considerações finais desta tese, e os trabalhos futuros.

2 CONCEITUAÇÃO TEÓRICA

Neste capítulo serão apresentados conceitos fundamentais sobre a plataforma Android, como estrutura das aplicações, componentes principais e seus funcionamentos, entre outras. Informações adicionais sobre termos técnicos estão disponíveis no Glossário desta tese.

2.1 A PLATAFORMA ANDROID

Em 2007, um grupo de 84 empresas de tecnologia, incluindo Google, Intel, Samsung e Sony, se uniram em um consórcio chamado OHA - *Open Handset Alliance* para criar padrões abertos de telefonia móvel (OHA, 2007). Quando foi anunciada a criação da OHA, também foi divulgada a plataforma Android, baseada no núcleo do Linux.

Partes essenciais da arquitetura do Android, conhecida como uma pilha de camadas, são apresentadas a seguir:

- **Kernel Linux:** O núcleo é a primeira camada de software que interage com a parte física do dispositivo. Entre as funções dessa camada estão o gerenciamento de processos, gerenciamento da memória disponível, segurança e rede. Para o desenvolvimento do sistema operacional Android foi utilizada uma versão de um núcleo (ou *kernel*) Linux. Todavia, essa versão não é genérica e pura. Devido às particularidades das empresas que produzem circuitos integrados e dispositivos, como celulares e tablets, são acrescentadas linhas de código específicos dos fabricantes e dos *drivers* (Khomh et al., 2012).

- **Camada de abstração de hardware (HAL):** Para permitir a comunicação entre as aplicações com o hardware do dispositivo é necessária uma camada de abstração (HAL). Essa camada fica mais próxima aos *drivers* e acima do núcleo (ou *kernel*) do sistema. Quando o desenvolvedor necessita chamar uma funcionalidade do hardware, o sistema operacional carrega módulos e bibliotecas referentes a esse componente.

- **Android *Run Time* (ART):** Devido à extensa variedade de aparelhos que utilizam o sistema Android e às restrições de quantidade de memória disponível, para executar aplicações é necessário um ambiente de emulação, como uma máquina virtual (Kaur e Sharma, 2014). Essa virtualização traz alguns prejuízos, incluindo gasto de energia e de memória do aparelho. Inicialmente, era utilizada a máquina virtual Dalvik. A partir da versão Android 4.4 KitKat, houve mudança do ambiente de execução para o ART (Khan e Shahzad, 2015). Esse ambiente foi desenvolvido para proporcionar maior velocidade aos aplicativos. Quando uma aplicação Android é criada, o desenvolvedor compila o código em um formato de *bytecode* intermediário, conhecido como formato DEX. Após o aplicativo ser carregado no dispositivo, o ART usa um processo conhecido como compilação *Ahead-of-Time* (AOT) para traduzir o *bytecode* para as instruções nativas (Lim et al., 2012). Essa camada é designada para rodar aplicações, especialmente devido às restrições de quantidade de memória disponível.

- **Bibliotecas do núcleo Android (*core libraries*):** O conjunto de bibliotecas denominadas *core libraries* disponibiliza funcionalidades referentes à programação Java, como acesso a banco de dados, renderização gráfica, comunicação inter-processos, gerenciamento da interface gráfica e outras.

- **Bibliotecas nativas C/C++:** grande número de componentes, como a camada HAL, foram desenvolvidos a partir de código nativo de linguagem C/C++. Por meio do Android NDK, que significa *Native Development Kit*, é possível incorporar linguagens nativas dentro de aplicações Android.

- **Java API framework:** Essa camada fornece serviços de níveis mais altos, como classes Java. Em relação ao desenvolvimento de aplicativos, as APIs disponíveis são agrupadas em módulos de serviços, como provedores de conteúdo, visão do sistema e gerenciadores. Esses módulos incluem funcionalidades para recursos de compartilhamento de dados entre aplicativos, desenvolvimento de interfaces gráficas e barras de notificação (Sihag et al., 2021).

- **Aplicativos do sistema:** Para garantir funcionalidades como envio e recebimento de mensagens SMS, navegador de internet, gerenciamento de contatos, calendário, calculadora e outros tipos de aplicações, o Android traz alguns aplicativos pré-instalados.

Após essa breve introdução à plataforma Android e a sua pilha de camadas, serão abordados os componentes de um arquivo compactado de aplicação Android (APK).

2.1.1 Estrutura de um APK

Geralmente os aplicativos para Android são disponibilizados em arquivos compactados, com a extensão *.apk*. O termo APK significa *Android application package*. A sua estrutura básica pode ser visualizada na Figura 2.1. Na Tabela 2.1 são apresentadas as informações sobre os diretórios e arquivos existentes em um APK.

Tabela 2.1: Componentes de um arquivo *.apk*

Componente	**Descrição**
Lib	Diretório que contém códigos nativos (extensão .so). Essa pasta traz subdiretórios para cada processador suportado, como armabi, arm64, x86, entre outros.
META-INF	Essa pasta tem informações de metadados, incluindo assinaturas e certificados que são utilizados para garantir e validar integridade.
CERT.RSA	Esse arquivo é o certificado do aplicativo, contendo a chave pública, geralmente um valor *hash* SHA1 ou SHA256. O APK deve ser assinado digitalmente pelo criador, para ser aceito pelas lojas de aplicativos.
CERT.SF	Lista de recursos utilizados com os respectivos *hashes* SHA-1.
MANIFEST.MF	Arquivo do Manifesto.
Res	Diretório de recursos do aplicativo, sendo dividido em vários subdiretórios, como *drawable*, *layout*, *mipmap* e *values*.
Assets	Diretório opcional, nesse diretório são armazenados recursos externos, que não são compilados. Exemplos são áudios, imagens e até *scripts* maliciosos.
AndroidManifest.xml	Arquivo de manifesto adicional do Android. O Manifesto contém informações sobre configurações do aplicativo, como permissões e componentes que serão utilizados. Durante a compilação, esse arquivo é transformado em código binário.
Classes.dex	Conjunto de arquivos que contém códigos compilados da aplicação. As classes estão no formato *bytecode*, que o ambiente de execução consegue entender e executar. Essas classes são importantes para engenharia reversa e análise estática do aplicativo.
Resources.arsc	Arquivo que faz a ligação entre o código (classes.dex) e o recurso (res), tendo informações do ID (identificação) do recurso correspondente. Como exemplos de recursos pre-compilados estão estilos, cores e textos.

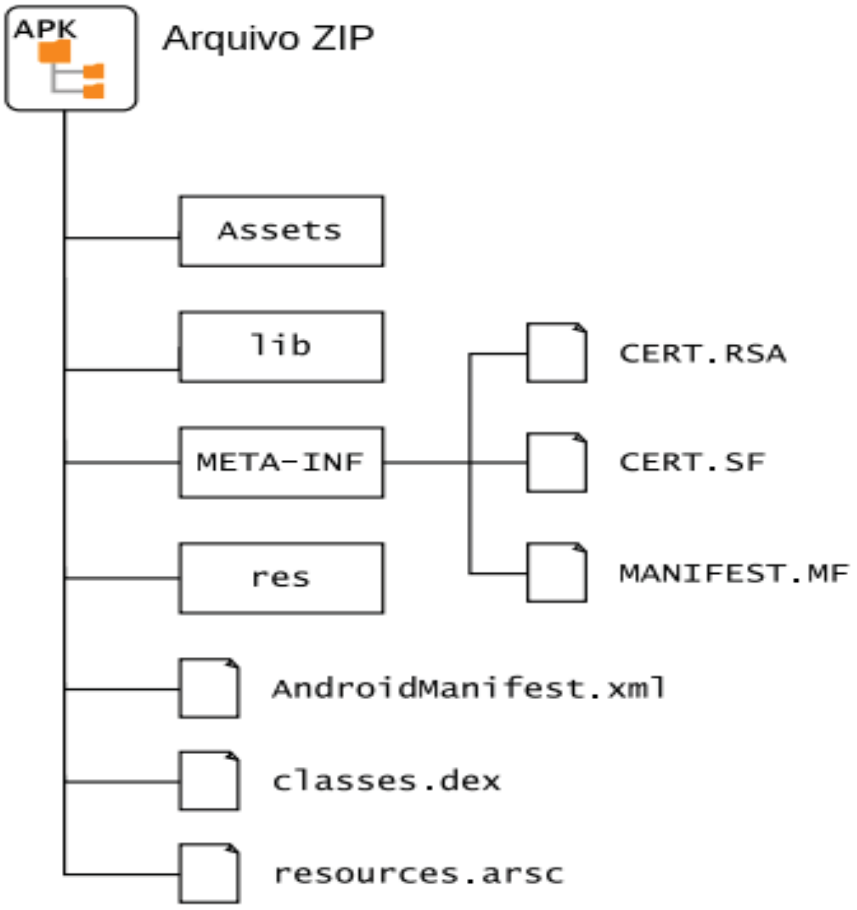


Figura 2.1: Estrutura de um arquivo APK. Fonte: Traduzido de Sihag et al. (2021)

A partir dos elementos mencionados anteriormente, para o desenvolvimento da aplicação são importantes os seguintes passos:

- **Passo 1**: os arquivos Java (compilados) são convertidos em *bytecode*. Durante a conversão, todos os recursos necessários são buscados na pasta Res.
- **Passo 2**: o *bytecode* pode ser executado na Máquina Virtual Dalvik (DVM). A saída dessa etapa é o arquivo Dex.
- **Passo 3**: o arquivo Dex é vinculado à pasta Resources.arsc para poder utilizar os pacotes, classes e funções.
- **Passo 4**: o arquivo Dex e os recursos são construídos para gerar o arquivo APK.
- **Passo 5**: o arquivo APK gerado é assinado, por meio de um certificado válido, para fornecer autenticação e integridade aos dados.
- **Passo 6**: o arquivo APK está pronto para publicação e para o processo de instalação. Durante a instalação, o APK é verificado e em caso de incompatibilidade de certificado, é exibido o erro na tela.

O formato APK tem sido utilizado desde o começo do sistema operacional. Desse modo, a maioria dos arquivos maliciosos disponíveis nos diretórios públicos estão nesse formato. Todavia, ressalta-se que a partir de agosto de 2022, aplicações publicadas na loja de aplicativos

do Google também podem estar em um novo formato: o AAB - *Android App Bundle* (Randhawa, 2022).

Além da estrutura básica de um arquivo compactado de aplicação Android, a próxima seção abordará quais são os componentes e como são utilizados no desenvolvimento de aplicativos para essa plataforma.

2.1.2 Intenções e componentes de aplicativos

Apesar de não ser visível ao usuário final, para desempenhar funções requeridas pelos jogos, comunicadores, aplicativos bancários e outras ferramentas disponíveis, as aplicações trocam mensagens com o sistema operacional. Para isso, o desenvolvimento de aplicativos é realizado com diversos tipos de elementos. Nesse cenário, componentes são como blocos de construção de uma aplicação para a plataforma Android. Os quatro tipos básicos são listados a seguir:

- **Atividades:** Uma atividade pode ser entendida como uma classe Java ou componente que preenche a tela do dispositivo (Mawlood-Yunis, 2022). Esses componentes são considerados processos ativos e visíveis aos usuários, sendo inicializados por intenções. Para serem utilizadas pela aplicação precisam estar declaradas no manifesto (*AndroidManifest.xml*).

- **Serviços:** O serviço é um recurso existente para manter um aplicativo em execução no segundo plano. Esse tipo de componente é utilizado para realizar operações de execução longa ou para processos remotos. Diferentemente de atividades, serviços não apresentam uma interface do usuário. Como exemplos, podem ser citados protetores de tela, detectores de notificações e serviços de acessibilidade.

- **Receptores de transmissão**: Transmissão, na plataforma Android, é um evento que pode ocorrer em diversas situações, como quando o dispositivo é iniciado, quando começa a ter a bateria carregada ou quando uma mensagem é recebida, por exemplo. Para os aplicativos é possível ter transmissões personalizadas, referentes a eventos que os interessam, como *download* de dados novos. Desse modo, os receptores de transmissão são componentes que servem para registrar eventos que ocorrem no sistema ou em aplicativos, sendo notificados quando acontecem.

- **Provedores de conteúdo**: Para armazenamento e gerenciamento seguro de dados entre aplicações foram criados os provedores de conteúdo. Por meio do provedor de conteúdo, outras aplicações podem consultar e modificar dados, caso o provedor responsável permita. Existem diversas formas para armazenamento persistente dos dados de aplicações Android, como sistemas de arquivos, banco de dados Sqlite, baseados em nuvem e outros locais dentro de dispositivos (MacLean et al., 2015).

Para ativar três tipos de componentes, atividades, serviços e receptores, são utilizadas mensagens assíncronas conhecidas como intenções. Essas mensagens são transferidas entre as duas aplicações e entre aplicações e o sistema operacional. Como exemplos de intenções, uma aplicação pode solicitar a abertura de um mapa ou a abertura da câmera do aparelho. Além disso, as intenções podem ser explícitas ou implícitas. A diferença é que as intenções implícitas não especificam o componente do aplicativo que desejam iniciar. Já as explícitas servem para achar um elemento diretamente pelo nome.

Diferentemente dos três componentes mencionados anteriormente, os provedores de conteúdo não são ativados por intenções. Esses elementos são gerenciados por classes de resolvedores de conteúdo (*ContentResolver*).

Além de serem essenciais para a criação de aplicações móveis, esses itens são amplamente investigados na análise estática de *malware*. Portanto, após serem abordados os componentes de aplicativos Android, nas próximas subseções serão comentadas configurações de segurança desse sistema operacional.

2.1.3 Modelo de segurança e vulnerabilidades

A plataforma Android foi desenvolvida com múltiplos níveis de segurança. Por meio de uma estrutura hierárquica, cada camada tem um método de proteção próprio, como controle de acesso, inspeção de permissões, uso de *sandbox*, assinatura digital e de criptografia (Sihag et al., 2021).

Como mencionado anteriormente, foi utilizado como base um *kernel* Linux para o desenvolvimento do sistema operacional Android, incorporando também o mecanismo de controle de acesso. Nesse modelo de segurança, o acesso a recursos e dados é restrito aos limites de autorização e de autenticação dos usuários. As camadas desse mecanismo podem ser visualizadas na Figura 2.2.

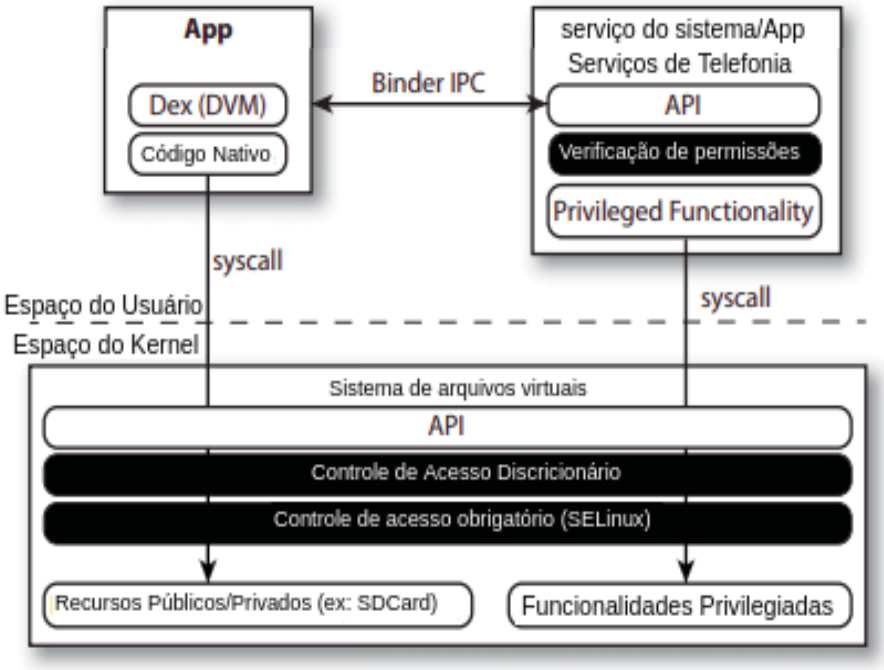


Figura 2.2: Modelo Padrão de Segurança do Sistema Android. Fonte: Traduzido de Backes et al. (2014)

Em relação a controles de acessos, o Android tem os controles de acesso obrigatório e discricionário. O primeiro gerencia as decisões sobre o módulo de segurança do Linux. Já o controle de acesso discricionário permite ao proprietário de determinado recurso gerenciar permissões associadas aos componentes.

Outro recurso de segurança é o uso de *sandbox*, que são ambientes de execução protegidos com memória e recursos próprios. Cada aplicação recebe um ID de usuário único,

dentro de uma *sandbox* separada. O ambiente esconde essa aplicação e suas configurações da interferência de outros aplicativos (Acharya et al., 2022a). Desse modo, a política de segurança objetiva que os aplicativos tenham nível de privilégio mínimo, evitando danos ao sistema e aos dados das contas de usuários (Backes et al., 2014).

Apesar de existirem protocolos e recursos para garantir a privacidade dos dados e a segurança do sistema, na literatura de análise de *malware* são encontrados diversos estudos sobre os modelos de segurança do sistema Android e seus problemas. Exemplos de pesquisas incluem os trabalhos de Liebergeld e Lange (2013), Faruki et al. (2014), Iqbal et al. (2018) e Acharya et al. (2022a).

Ainda que atualizações e correções de falhas sejam disponibilizadas periodicamente, atacantes e criadores de códigos maliciosos encontram formas de ganhar privilégios e acesso a dispositivos vulneráveis. Alguns problemas de segurança críticos são listados a seguir:

Falha na implementação do mecanismo de segurança: assim como no desenvolvimento de outros tipos de software, execução de testes rigorosos são essenciais para detectar antecipadamente falhas de segurança nas versões do sistema operacional Android. Se as vulnerabilidades permanecerem não detectadas, podem ser exploradas por atacantes para obtenção de credenciais de administradores do sistema.

Vulnerabilidades nos códigos-fonte de aplicações: além de pontos críticos e falhas de segurança no sistema operacional, são comuns *bugs* em códigos-fonte nativos de aplicativos. Por meio de técnicas de regeneração de código, atacantes podem utilizar essas falhas para adicionar códigos maliciosos aos APKs (Acharya et al., 2022a).

Configuração incorreta: arquivos de configuração de aplicativos também podem ser facilmente explorados para ataques e roubos de informações do usuário.

A seguir são apresentados conceitos fundamentais sobre *malware*, incluindo categorias, ativação e formas de análise de aplicativos maliciosos.

Depois dessa breve introdução sobre a plataforma Android, nas próximas seções são apresentadas informações sobre *malware* em geral e *malware* focados nessa plataforma.

2.2 *MALWARE*: NOMENCLATURA E CLASSIFICAÇÕES EM GERAL

A palavra *malware* refere-se a softwares maliciosos, também conhecidos como *malcodes* ou códigos maliciosos. O cenário da propagação de *malware* tem se expandido a cada ano, devido ao desenvolvimento de técnicas avançadas de escrita e modificação de códigos maliciosos existentes. Uma família de *malware* contém exemplares (variantes) que implementam as mesmas funcionalidades ou que possuem comportamentos ou estruturas similares.

Devido à grande quantidade de variantes que têm sido criadas automaticamente, os quais contornam mecanismos de detecção de padrões e assinaturas, a análise manual tem se tornado um gargalo significativo no controle das pragas virtuais (Awad e Sayre, 2016). Entre os tipos de *malware* amplamente conhecidos, destacam-se (Elhadi et al., 2012), (Wang, 2006):

- **Vírus** são programas que se auto-replicam dentro de um *host*, se anexando a documentos e arquivos, que os carregam adiante.
- **Worms** também se auto-replicam por meio de uma rede.
- **Trojans** ou cavalos-de-tróia se passam por softwares úteis, mas trazem códigos maliciosos que atacam o sistema ou roubam dados dos usuários.
- *Malware* chamados de **backdoors** subvertem as políticas de segurança para permitir acesso remoto a atacantes.

- **Spywares** são códigos maliciosos utilizados para transmitir dados privativos de usuários para uma entidade externa. Geralmente, são anexados a softwares genuínos.

- **Rootkits** são conjuntos de ferramentas utilizadas por atacantes para ganhar privilégios administrativos em um *host*.

- **Botnets** são programas remotamente controlados, que permitem ao atacante realizar tarefas de forma automatizada em máquinas "zumbis".

- **Adware** também é chamado de software suportado por publicidade. Sua funcionalidade é exibir ou fazer ou baixar anúncios para um computador, após a instalação de software ou aplicativo mal-intencionado (Mathur e Hiranwal, 2013) .

Apesar de categorias bem definidas com relação a comportamentos desempenhados por *malware*, o esquema de nomes ainda é um problema complexo. Analistas de *malware* e empresas de antivírus nunca concordaram em um único esquema de nomes para *malware* (Maggi et al., 2011). Essa falta de padronização acarreta diversas inconsistências e dificuldades nas pesquisas de variantes e de suas famílias.

Em relação às formas de obtenção de *malware* para pesquisa, além do uso de *honeypots*, existem diretórios públicos. Os *datasets* mais utilizados na literatura são apresentados a seguir.

2.3 ABORDAGENS DE CLASSIFICAÇÃO DE CÓDIGOS MALICIOSOS

As abordagens de classificação de *malware* podem ser divididas em baseadas em algoritmos ou baseadas em assinaturas (Abusitta et al., 2021).

Por meio de técnicas de análise, que serão abordadas a seguir, são criadas assinaturas baseadas em *host* ou em rede. Assinaturas baseadas em *host*, também chamadas de indicadores, são empregadas para detectar código malicioso em computadores infectados. Esses indicadores, geralmente, identificam arquivos criados ou modificados por *malware* ou mudanças específicas que esses programas maliciosos fazem no registro do sistema operacional (Sikorski e Honig, 2012).

Já as assinaturas de rede são utilizadas para detecção de código malicioso mediante monitoramento de tráfego de rede. Assinaturas de rede podem ser criadas sem análise de *malware*. Entretanto, verifica-se que as assinaturas criadas após análise oferecem taxas de detecção mais altas e menos falsos positivos (Sikorski e Honig, 2012).

Na segunda abordagem de classificação de exemplares, a baseada em algoritmos, encontram-se técnicas de aprendizado de máquina, as quais incluem:

- **Aprendizado de máquina Supervisionado:** grupo de algoritmos que aprendem com dados rotulados de exemplos já classificados, utilizando um atributo denominado de Classe. Esses métodos podem avaliar sua precisão a partir dos dados de treinamento (Backer, 1995).

- **Aprendizado de máquina Não-supervisionado:** diferentemente da anterior, na abordagem não-supervisionada não há classe associada aos exemplos. Desse modo, o algoritmo analisa os dados fornecidos e tenta determinar se alguns deles podem ser agrupados de acordo com algum padrão, formando agrupamentos ou *clusters* (Cheeseman et al., 1996). Esses tipos de algoritmos são o foco desta pesquisa.

- **Aprendizado de máquina Semi-supervisionado:** métodos mais recentes, algoritmos semi-supervisionados representam a junção dos dois anteriores, possibilitando a redução da necessidade de dados rotulados, devido a situações em que apenas um pequeno conjunto de amostras rotuladas está disponível (Matsubara, 2004) .

Em relação a estratégia de classificação utilizada, podem ser citados os seguintes tipos (Abusitta et al., 2021):

- **Análise de similaridade:** nesse tipo de abordagem são investigadas partes de código malicioso, com objetivo de detectar funções novas, ou seja, trechos de código não vistos em outras amostras do diretório de exemplares. Um exemplo dessa técnica é a detecção de clonagem de código.

- **Análise de famílias:** essa estratégia funciona supondo que *malware* da mesma família são semelhantes entre si em termos de funcionalidade. Todavia, técnicas de anti-análise, incluindo ofuscação, empacotamento, polimorfismo e metamorfismo, são empregadas para ocultar essa semelhança e dificultar a detecção e a classificação.

- **Análise de variantes:** como amostras maliciosas podem ser variantes de *malware* já conhecidos, por meio de análise de variantes os analistas podem entender qual código foi evoluído ao longo do tempo.

2.3.1 Evolução da Camuflagem dos códigos maliciosos

Com o avanço das técnicas de escrita e reescrita de *malware*, podem ser utilizados métodos como empacotamento, metamorfismo e polimorfismo de códigos. Empacotadores são mecanismos usados para compressão e criptografia para esconder códigos maliciosos. Durante a execução do *malware*, mecanismos avançados podem restaurar o códigos e sub-rotinas maliciosas na memória da máquina e executá-los (Ling e Sani, 2017). *Malware* polimórficos podem também utilizar algoritmos de criptografia. A cada vez que o programa malicioso é executado, o código é decifrado e armazenado na memória (Ling e Sani, 2017). Entretanto, a cada execução é produzida uma assinatura diferente. Outro mecanismo presente em códigos maliciosos para burlar a detecção, são as diferentes técnicas utilizadas por *malware* metamórficos para adaptar seus códigos em cada replicação. Porém, apresentam funcionalidades maliciosas iguais ou semelhantes. Geralmente, os motores metamórficos são projetados para serem pequenos visando contornar a detecção (Chouchane e Lakhotia, 2006)

Algumas técnicas de ofuscação são listadas a seguir:

- Inserção de código morto ou "lixo"
- Substituição de instruções por equivalentes
- Armazenamento em registros trocados/diferentes
- Permutação de códigos e sub-rotinas
- Mutação de códigos de *hosts*

Como novas variantes de *malware* com camuflagens diferentes são liberadas diariamente na Internet, torna-se essencial o uso de métodos de agrupamento para facilitar o estudo de grandes conjuntos, os quais podem conter milhares de amostras. Desse modo, a próxima seção apresenta uma curta introdução sobre o assunto.

2.4 FORMAS DE DETECÇÃO DE MALICIOSIDADE

As principais técnicas de detecção de *malware* são as baseadas em assinaturas, baseada em anomalias de comportamento e técnicas que combinam as duas, gerando abordagens híbridas (Singla et al., 2015).

- **Métodos baseados em assinatura:** Esquemas baseados em assinatura são os mais utilizados em sistemas antivírus comerciais, devido à baixa taxa de falsos positivos e baixa complexidade computacional. São utilizados bancos de dados de assinaturas, que são padrões de sequências de *bytes* específicos associados a arquivos maliciosos (Ling e Sani, 2017). Entretanto, esse método de detecção é falho em relação a *malware* que ainda não foram catalogados, como ataques dia-zero e também contra *malware* com técnicas de ofuscação de código.

- **Métodos baseados em anomalias:** Nos métodos baseados em comportamento, são utilizadas listas de comportamentos maliciosos na comparação de indicadores de *malware*. Os antivírus buscam por comportamentos anormais ou suspeitos de arquivos e processos que estão sendo monitorados. Uma das maiores vantagens desse esquema é a habilidade de detectar ataques de dia-zero. Todavia, como desvantagens podem ser citadas a alta taxa de falsos positivos e maior complexidade computacional.

- **Métodos baseados em heurística:** Nesses métodos são utilizadas heurísticas para análise de características consideradas suspeitas em arquivos. Heurísticas são um conjunto de regras capazes de diferenciar o comportamento de um código malicioso de um comportamento de um arquivo genuíno. Nesse tipo de análise são utilizadas técnicas de aprendizagem de máquina e de mineração de dados para aprender a diferenciar quais características ou comportamentos melhor representam o arquivo analisado.

Na próxima seção serão abordadas medidas avançadas encontradas em *malware* evoluídos, as quais dificultam tarefas de analistas de Segurança, tanto em detecção, remoção quanto em classificação de códigos maliciosos.

2.5 CONJUNTOS PÚBLICOS DE AMOSTRAS DE *APKS* PARA *DOWNLOAD*

Devido ao potencial de danos das aplicações maliciosas, diversos repositórios de exemplares de *malware* para pesquisa surgiram nos últimos anos. A Tabela 2.2 fornece informações sobre alguns conjuntos de *apk* maliciosos existentes.

Tabela 2.2: Conjuntos de amostras maliciosas focadas no sistema operacional Android. Os sites do M0Droid e AMD Project não estavam disponíveis enquanto este documento estava em progresso.

Nome	**Coleta**	**Amostras**	**Famílias**	**URL**
Malgenome	2011-2015	1.260	49	malgenomeproject.org
Drebin	2010-2012	5.560	179	sec.cs.tu-bs.de/ danarp/drebin
M0Droid	2011-2013	193	-	cyberscientist.org/m0droid-dataset
Koodous	2017	> 100.000	-	koodous.com
VirusShare	2016	34.140.004	-	virusshare.com
Androzoo	2016	10 mi	76 000	androzoo.uni.lu
PRAGuard	2015	10.479	-	pralab.diee.unica.it /en/AndroidPRAGuardDataset
AAGM Dataset	2017	1.900	12	unb.ca/cic/datasets/android-adware.html

AMD Project	2017	4354	42	amd.arguslab.org
Kharon Dataset	2016	60	19	kharon.gforge.inria.fr/dataset
Contagio	2011-2013	237	-	contagiodump.blogspot.com
Piggybacking	2011-2016	1.136	29	https://github.com/serval-snt-uni-lu/Piggybacking
RmvDroid	2014-2018	9.133	56	https://doi.org/10.5281/zenodo.2593596
AndMal2017	2015-2017	426	42	https://www.unb.ca/cic/datasets/andmal2017.html
AndMal2020	2006-2020	400.000	191	https://www.unb.ca/cic/datasets/andmal2020.html
MalDroid20	2017-2018	17.341	-	https://www.unb.ca/cic/datasets/maldroid-2020.html

Em relação ao número de variantes disponíveis, os menores conjuntos de dados são Kharon (60 amostras), M0Droid (193 amostras) e Contagio (237 amostras). Os dois maiores são o Androzoo, com mais de 10 milhões de amostras, e o Koodous, como mais de 100 mil exemplares.

O primeiro conjunto público de amostras de *malware* Android, lançado em 2011, e um dos mais populares na literatura é o Malgenome. São disponibilizados 1.260 variantes de 49 famílias de *malware* Android coletados entre agosto de 2010 e outubro de 2011.

Além disso, relata-se que o conjunto de dados PRAGuard contém modificações e ofuscação das variantes presentes nos repositórios Malgenome e Contagio.

O conjunto de dados Drebin, disponível publicamente em 2014, possui 5.560 aplicativos maliciosos de 179 famílias distintas, com período de coleta entre 2010 e 2012.

As informações sobre o conjunto de dados do AAGM indicam que 250 *malware* do tipo Adware, 150 *malware* genéricos e 1.500 aplicativos benignos pertencem ao conjunto. Além disso, as amostras correspondem a variantes de 12 famílias.

Wei et al. (2017) criaram a base de *malware* AMD, fornecendo informações como características da família, incluindo relatórios de comportamento de variantes.

2.6 FORMAS DE ATIVAÇÃO E DE INSTALAÇÃO DE *MALWARE*

De acordo com as características de ativação e de instalação do aplicativo malicioso no dispositivo, as categorias principais são Reempacotamento, Ataque de atualização e *Drive-by Download*.

- **Reempacotamento:** Uma das técnicas populares entre os atacantes e criadores de *malware* é o reempacotamento. Em geral, os desenvolvedores localizam aplicativos populares, os desmontam, incluem *scripts* maliciosos e em seguida, os remontam e os enviam para loja de aplicativos oficiais ou alternativas (Zhou e Jiang, 2012a). A Figura 2.3 apresenta um exemplo de *malware* para Android utilizando essa técnica.

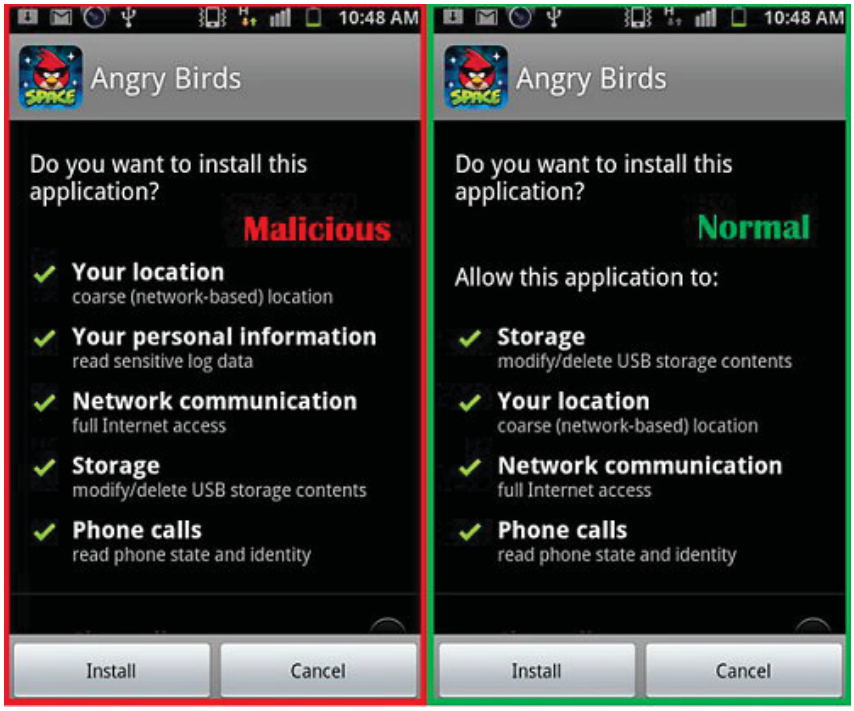


Figura 2.3: Exemplo do jogo Angry Birds reempacotado com *malware*. Fonte: Joo e Hwang (2012)

- **Ataque de atualização:** Diferentemente da primeira técnica, onde os autores colocam um componente maliciosos inteiro escondido na aplicação, nesse tipo de ataque apenas uma parte do código é colocada no arquivo compactado para instalação. Desse modo, o *malware* faz *download* do restante dos componentes maliciosos em tempo de execução, dificultando assim a detecção inicial da maliciosidade. Na Figura 2.4 são apresentados exemplos de *malware* utilizando essa técnica.

Figura 2.4: Exemplo de código e de janela relacionado a ataques de atualização das famílias Anserver, BaseBridge e DroidKungFuUpdate. Fonte: Adaptado e traduzido de Zhou e Jiang (2012a)

- ***Drive-by Download*:** O terceiro tipo de ataque é utilizado por aplicativos e páginas *Web* que incitam o usuário a fazer *download* de outras aplicações, através de *QRCodes*

e *links* em anúncios. As promessas para incentivar os *downloads* incluem melhorias na vida útil da bateria do dispositivo e proteção extra para aplicativos de banco (Jiang et al., 2013).

Em relação à ativação, alguns *malware* Android utilizam eventos do sistema para iniciarem atividades no dispositivo, a partir de eventos de *boot* e de recebimento de mensagens SMS.

2.6.1 Famílias de *malware* Android

O estudo das famílias de *malware*, tanto para sistemas móveis, quanto para outros tipos de sistemas operacionais, é parte essencial do desenvolvimento de técnicas de segurança, correções de vulnerabilidades, soluções para mitigação de dados e prevenção de ataques. A Tabela 2.3 disponibiliza informações sobre famílias populares de *malware* relacionadas à plataforma Android.

Tabela 2.3: Famílias de *malware* Android e suas descrições

Nome/Família	**Descrição/ Como age**
Admogo	Trata-se de um tipo de *Adware*, ou seja, um tipo de software indesejado que causa excesso de anúncios e janelas pop-ups no dispositivo ou no navegador *Web* (Medina, 2016).
Adwo	Também é do tipo Adware, exibindo anúncios indesejados, podendo mostrar notificações e é considerado invasivo à privacidade (Xie et al., 2019).
Airpush	Família de *malware* também focada em anúncios e notificações excessivas. Eles também podem fazer alterações nos favoritos do navegador ou nos ícones da tela inicial.
Android.spy.277	Rouba informações, cria ícones na área de trabalho, apresenta anúncios falando de perigos de bateria com temperatura alta. Também envia informações do usuário para servidores remotos.
Android Defender	Promete ser um *software* antivírus, mas é um *malware*. Depois de uma falsa varredura a procura de ameças no dispositivo do usuário, ele fornece *links* de soluções de vacinas e de remoções de *trojans* encontrados.
AnserverBot	Provavelmente é uma evolução da família BaseBridge. Possui técnicas avançadas de evasão. Através de um *backdoor* no dispositivo Android, esse tipo de *malware* consegue roubar informações pessoais e envia-las para um servidor remoto (Spreitzenbarth e Freiling, 2012).
Anydown	É considerada do grupo de famílias virais do Android. As variantes dessa família possuem grande parte de métodos em comum, sendo mais de 280 funcionalidades compartilhadas entre as amostras (Suarez-Tangil e Stringhini, 2018).

Artemis	O rótulo Artemis foi padronizado pela empresa Mcafee (Martín et al., 2019b), tornando-a uma das mais numerosas do Android. Geralmente é utilizado para *trojans Adware*.
AvForAndroid	Família do grupo de *Scareware* (Fiky et al., 2021). Utiliza comunicação com servidores do tipo C&C para enviar mensages SMS, abrir sites e fazer ligações telefônicas
AvPass	Também é do grupo de *Scareware*, sendo a única que interage com o antivírus do dispositivo, prejudicando o sistema de detecção e tornando-o ainda mais vulnerável (Gautam e Rahimi, 2023).
Deng	Classificada como um *Riskware* (Fiky et al., 2021). Essa família é distribuída com aplicativos legítimos, porém após instalação redireciona o usuário para *sites* falsos. Pede privilégios de administrador do sistema.
Domob	Instala um *plugin* de alto risco e rouba informação pessoal (Medina, 2016). Como é considerada do tipo *Adware*, apresenta anúncios excessivos para o usuário.
Dowgin	Essa família de *Adware* infecta navegadores de internet. Além de apresentar grande quantidade de anúncios, redireciona o usuário para *links* patrocinados, instala barras de ferramentas e faz outras modificações nos navegadores (Ullah et al., 2022).
DroidDream	Utiliza meios maliciosos para obter privilégios de administrador, acessar informações pessoais presente no dispositivo, depois as envia a um servidor remoto (Y. Kim e Chan, 2016).
Droidkungfu	Utiliza recursos para obter privilégios de administrador do sistema. Inicialmente foi desenvolvida com foco em usuários chineses, porém teve distribuição extensa e hoje encontra-se grande número de variantes em aplicações como jogos (Medina, 2016)
FakeApp	Família do grupo de *Scareware* (Fiky et al., 2021)
FakeApp AL	Família do grupo de *Scareware* (Fiky et al., 2021)
FakeAV	Família do tipo *Scareware*. Como o próprio nome já indica, se passa por uma solução antivírus falsa, vendendo para o usuário aplicativos para remover ameaças (Korine e Hendler, 2021).
FakeJobOffer	Aplicativo que oferece vagas em uma multinacional indiana, pedindo um depósito bancário para mais informações. O *malware* apresenta uma carta falsa para o usuário cada vez em que o dispositivo é ligado (Security, 2013).
FaketaoBao	É da família de*Scareware* (Taheri et al., 2019). Taobao é um portal de compras popular chinês. Como o nome sugere, esse *malware* é um Taobao falso.
Feiwo	Família de *Adware*. Envia informações pessoais do usuário para servidor remoto.

Genpua	Poucas informações sobre esse rótulo de família de *malware* na internet. Sabe-se que o nome vem de uma abreviação do termo PUA (*Potentially unwanted applications*) - aplicações potencialmente indesejáveis (Fauskrud, 2019).
Geyser	Família de *Adware*. Esse *malware* compartilha dados da localização geográfica do dispositivo.
Ginmaster	*Malware* que pode fazer alterações inclusive em cartões de memória do dispositivo do usuário. Uma família extensa, que já teve mais de grandes alterações na estrutura, desde quando foi identificada, em 2011 (Yu, 2013).
Inmobi	Família de *Adware*. Faz instalações de barras de ferramentas e outros itens em navegadores de Internet.
Jiagu	Classificada como *Riskware*, podendo apresentar comportamentos de "clicar"em anúncios patrocinados (Fiky et al., 2021).
Kuguo	*Malware* que apresenta comportamento e características similares às famílias Youmi e Dowgin, frequentemente causando problemas de rotulação de variantes (Otani et al., 2020).
Leadbolt	Família de *Adware*, frequentemente confundida com a família Plankton (Lin et al., 2022).
MisoSMS	*Malware* que tem comportamento de enviar mensagens SMS do dispositivo usuário para atacantes (Pieterse e Burke, 2015a).
NickySpy	Categorizada como *Spyware*, grava as ligações telefônicas do usuário em cartão de memória para depois envia-las a servidores remotos (Dini et al., 2012).
Not Compatible	A família Not Compatible apresenta comportamentos de *Botnet*, utilizando "túneis", ou seja, passagens seguras e separadas para enviar informações para servidores maliciosos (Vidas et al., 2014).
Penetho	*Malware* que vem acoplado a ferramentas do tipo *Hacktool*, como aplicativos quem podem ser utilizados para roubo de informações e credenciais de redes Wi-fi (Lashkari et al., 2018).
PJapps	Família de *trojans* com capacidade de *backdoor*, rodando atividades maliciosas em segundo plano no dispostivo do usuário (Burguera et al., 2011).
Plankton	Considerada como *Trojan_SMS*, utiliza técnicas de ataque de atualização (Shakya e Dave, 2022) (Fiky et al., 2021).
Pletor	*Malware* que apresenta características de *ransomware* (Sharma et al., 2021).
Revmob	Família relacionada a redes maliciosas de anúncios, apresentando ferramentas que emulam cliques em *links* específicos para monetização (Cho et al., 2015).
Skymobi	*Malware* que apresenta características de *Riskware* (Fiky et al., 2021).

Smspay	*Malware* que apresenta características de *Riskware* (Fiky et al., 2021).
TigerBot	Rouba informações como localização GPS, encerra processos rodando, lê e exclui mensagens SMS maliciosas (S. Josh e Joshi, 2015).
Umeng	Poucas informações sobre esse rótulo de família de *malware* na internet. Malware que apresenta características de *Adware* (Gajrani et al., 2020). Similiar a famílias como Woobo.
Virus Shield	Família do grupo de *Scareware* (Fiky et al., 2021).
Waps	Família de *Adware*, frequentemente confundida com Ginmaster e Wooboo.
Wooboo	Família de *Adware*, frequentemente confundida com Ginmaster e Waps.
Wroba	Família de *malware* bancário. Utiliza diversas técnicas de ofuscação e criptografia. Rouba credenciais do dispositivo do usuário (Kadir et al., 2016)
Youmi	Família extensa de *malware*, apresenta comportamento e características similares às famílias Kuguo e Dowgin, frequentemente causando problemas de rotulação de variantes (Fiky et al., 2021) (Otani et al., 2020).
Zitmo	*Trojan* que monitora as mensagens SMS a procura de códigos usados para completar transações bancárias (Kadir et al., 2016).

2.7 FORMAS DE ANÁLISE DE *MALWARE*

O objetivo do processo de análise de *malware* é determinar as funcionalidades dos códigos investigados, extraindo a maior quantidade possível de informações (Caldas, 2016). A partir da análise e da descoberta de sua estrutura, comportamento e de seus pontos fracos, pode-se desenvolver "vacinas".

Existem basicamente dois tipos de análise de *malware*: análise estática e análise dinâmica. Destaca-se ainda que ambas podem ainda ser categorizadas em básicas ou avançadas, as quais serão detalhadas nas próximas seções.

2.7.1 Análise Estática

Análise estática consiste no exame do arquivo executável sem visualizar as instruções propriamente ditas. A análise estática básica pode confirmar se um arquivo realmente é malicioso e, em alguns casos, disponibiliza certos dados que permitem produzir assinaturas de rede. Desse modo, essa análise no modo básico é rápida, porém pode ser ineficiente contra *malware* sofisticados, resultando em perdas de comportamentos importantes.

A análise estática avançada envolve engenharia reversa do *malware* através de um *disassembler*. Como as instruções são executadas pelo processador, esse tipo de análise estática consegue revelar o que o programa faz. Portanto, para realizá-la necessita-se de conhecimento especializado de *disassembly*, construção de código e conceitos de sistemas operacionais (Sikorski e Honig, 2012).

Na Tabela 2.4 são apresentadas algumas das principais técnicas de Análise Estática.

Tabela 2.4: Técnicas de Análise Estática

Técnicas	Descrição
Análise de Fluxo de Controle	Conhecido como CFG, consiste em um diagrama de blocos que demonstra possíveis fluxos entre instruções de um código. Gera-se um grafo que funciona como uma assinatura do código malicioso.
Análise do Fluxo de Dados	Necessita da criação do diagrama CFG. Fornece informações de todas as variáveis encontradas em um programa.
Análise de Dependência dos Dados	Essa técnica é utilizada por analistas e também por criadores de *malware*, permitindo a criação de códigos maliciosos metamórficos. A dependência dos dados existe se há um fluxo necessário entre um conjunto de instruções, impedindo que elas sejam ordenadas de forma totalmente aleatória.
Fatiamento	Método para traçar as computações realizadas em uma determinada variável do código.
Fingerprinting	Envolve operações em nível de arquivo, como calcular *hashs* criptográficos do arquivo binário (MD5, SHA-1, etc.), com o objetivo de identificar e distinguir o *malware* de outros arquivos, e verificar que o arquivo não foi modificado.
Extração de *strings*	Programas normalmente exibem mensagens de texto, que podem acabar inseridas no código binário como cadeias de caracteres de texto. Examinar tais cadeias de caracteres (*strings*) permite ao investigador tecer conclusões a respeito do *malware*, orientando os próximos passos da análise.
Detecção de *Packers*	Atualmente a maior parte dos *malware* são encontrados de forma compactada ou encriptada. Isso pode ser feito com o uso de *packers*, programas feitos para disfarçar o *malware* e dificultar sua análise pelos investigadores.
Disassembly	A maior parte da análise estática se dá através da engenharia reversa do código, por meio do *disassembly* de seu binário. É feita utilizando ferramentas capazes de converter o código de máquina em linguagem *assembly*. Com o código *assembly* reconstruído, o investigador pode inspecionar a lógica do programa, descobrindo sua função e como ele trabalha.
Descompiladores	Ferramentas cujo objetivo é reverter o binário em código fonte, fazendo o caminho reverso do processo de compilação. Tal recuperação não é possível na maioria das arquiteturas e linguagens, mas pode ser útil em casos específicos.

Tabela 2.4: Técnicas de Análise Estática - Adaptado de Caldas (2016)

2.7.2 Análise Dinâmica

A análise dinâmica básica consiste na execução do programa malicioso em um ambiente controlado, observar seu comportamento no sistema para poder remover a infecção e produzir assinaturas. Do mesmo modo que a análise estática básica, esse tipo de análise pode ser feito por pessoas sem conhecimento profundo em programação. Todavia não será efetiva contra todos os tipos de *malware*, podendo também perder detalhes importantes de funcionalidades sofisticadas.

Na análise dinâmica avançada é utilizado um depurador para examinar o fluxo de execução do programa para uma determinada entrada. Esse tipo de análise tem vantagem quando comparada à análise

estática em relação a técnicas de ofuscação que alguns *malware* realizam. Uma desvantagem, porém, é que alguns programas maliciosos são programados para alterar seus comportamentos quando percebem que estão sendo executados em ambientes controlados (Caldas, 2016).

2.7.3 Análise Híbrida, Automática e baseada em Nuvem

Realizar análise de *malware* de modo automático trata-se de utilizar técnicas com pouca ou nenhuma intervenção humana. Os dois principais métodos são: monitoramento de ações em tempo de execução e a comparação do estado do sistema operacional antes e depois da execução da amostra de *malware* (Willems et al., 2007).

O primeiro método mencionado captura dados relacionados à execução, normalmente mediante *hooks* (ganchos) como injeções de DLL (Sikorski e Honig, 2012). Já a comparação dos estados do sistema, apesar de não detectar mudanças como chamadas de APIs, gera uma visão geral do que programa malicioso faz, como arquivos que são e/ou deletados, entre outros. Já na análise híbrida, técnicas da análise estática e da análise dinâmica são combinadas a fim de contornar as limitações de cada uma.

Na análise de aplicativos maliciosos, devido a limitações de bateria e processamento, podem ser utilizados emuladores e réplicas de aparelhos em ambientes de processamento e armazenamento em nuvem. São emulados os processos rodando no telefone móvel, retornando ao analista os resultados processados. Um dos benefícios dessa metodologia é a possibilidade de usar diversos detectores simultaneamente, por haver maior liberdade de espaço e processamento nos servidores de nuvem (Mohite e Sonar, 2014).

A seguir são abordadas outras atividades e técnicas utilizadas na análise dos dados coletados.

2.7.4 Extração de atributos e representação de dados

Na análise e modelagem de informações do mundo real na computação, atributos ou características de objetos são armazenados em bases e arquivos. Além disso, *features* ou características distintas são fatores decisivos para classificação, agrupamentos e outros tipos de tarefas. Como exemplos de características utilizadas para descrição de humanos, encontram-se cor de olhos, altura, peso, idade, gênero, entre outras. Já na área de *malware*, características interessantes incluem extensão do arquivo e tipo do arquivo malicioso, tamanho do executável, conjunto de bibliotecas utilizadas e ícones.

No contexto de análise de códigos maliciosos, os atributos podem ser extraídos e armazenados de diversas formas. Todavia, a qualidade do método de extração de atributos implica diretamente na acurácia do algoritmo de classificação de *malware*. Portanto, devem ser selecionadas características que representem a estrutura e o comportamento da variante analisada (Caldas, 2016).

A Figura 2.5 apresenta exemplos de métodos relacionados à extração de atributos, representação de dados e seleção de características.

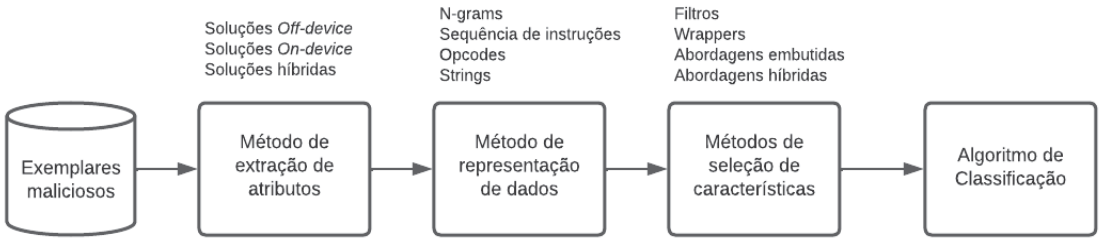


Figura 2.5: Procedimentos relacionados a atributos

Algumas técnicas de extração de atributos são apresentadas a seguir:

- ***Opcodes***: abreviação de códigos de operação, também conhecidos como sílabas de instrução, pacote de instruções ou *opstring*. São códigos que fazem parte de linguagem de máquina, os quais especificam quais operações serão executadas.

- **Extração de *Strings***: por meio da análise do texto plano presente nos arquivos binários das aplicações, investiga-se dados como nomes de classes externas, textos de janelas e de outros itens gráficos (Killam et al., 2016).
- **Sequência de Instruções**: Assim como nos softwares genuínos, nos softwares maliciosos são executadas diversas instruções. Desse modo, uma determinada sequência extraída pode ser utilizada para representar comportamentos maliciosos de *malware*. A sequência ou parte da *string* é comparada com listas de comportamentos maliciosos, como no processo da detecção da assinatura do *malware*.
- **Sequência de Chamadas de Funções**: Binários podem ser razoavelmente bem identificados como normais e maliciosos, observando apenas os nomes das funções e chamadas que aparecem em seus códigos (Schmidt et al., 2009).
- **Grafos de fluxo de controle - CFG**: o CFG de um programa mostra todos os caminhos que podem ser usados durante a execução de um programa (Alam et al., 2015). Ressalta-se que a complexidade computacional aumenta proporcionalmente ao tamanho do grafo gerado.
- **Grafos de fluxo de dados**: Nesse método, são extraídos fluxos de dados de todo o sistema, os quais são os fluxos de dados entre diferentes entidades do sistema, que ocorreram dentro de um período de tempo específico (Wüchner et al., 2014).
- **Chamadas a API perigosas:** existem técnicas que detectam chamadas a determinadas API e solicitações de permissões perigosas. Desse modo, se o programa fizer pelo menos uma dessas chamadas, será considerado desconfiável (Skovoroda e Gamayunov, 2015).
- **Traços de rede e de comunicação:** A maioria dos *malware* precisa de conexão de rede para poder se espalhar e também compartilhar dados de suas atividades maliciosas. A partir de registros de monitoramento de rede, traços podem ser facilmente extraídos para detecção de comportamento de *malware* (Perdisci et al., 2010).

Ressalta-se que dependendo da complexidade e da natureza das informações das amostras analisadas, necessita-se de métodos de extração de atributos mais apropriados. Exemplos de representação de dados e de características são disponibilizados na Tabela 2.5.

Tabela 2.5: Exemplos de representação de dados e extração de características

Tipo	**Descrição**
N-grams	Os programas são divididos em *strings/bytes* podem ser divididos em instruções de montagem, linhas de código e assim por diante (Karim et al., 2005a). Exemplos: *byte code* 2-gram e opcode 3-gram.
Frequência Binária	O vetor binário é direcionado ao tamanho do dicionário de termos e características, no qual: 0 = ausência do termo, 1 = presença do elemento/características na código da amostra. Exemplo: vetor binário de permissões do aplicativo
Extração da frequência da característica	Nesse tipo de extração, o vetor contém o número de ocorrências dos termos do dicionário, não apenas a ausência e presença da característica na amostra (Imran et al., 2017)
Fatores de ponderação de frequência	Além de calcular a frequência de termos, métodos como TF-IDF usam pesos para determinar a importância do atributo no conjunto de características.
Modelo oculto de Markov (HMM)	Apesar de não ser considerado apenas uma forma de extração de características, os HMMs são modelos estocásticos que representam estados, tendo resultados satisfatórios na detecção de *malware*.

Em relação à literatura estudada, no *survey* de Siddiqui et al. (2008) destacou-se que as principais formas e atributos utilizados para análise de *malware* são: *n-grams*, instruções, características híbridas e chamadas de sistema/API. Já no estudo de Ye et al. (2017), em relação à análise estática, os principais métodos encontrados na literatura foram: chamadas de APIs do Windows, *N-grams*, Sequências de caracteres (*strings*), códigos operacionais (*opcodes*) e CFG (grafos de controle de fluxo). Outras características estáticas para representação de arquivos utilizadas são propriedades do arquivo, informação de recursos do arquivo e tabela de exportação.

2.7.5 Seleção de características

Devido ao alto número de dimensões dos dados e de variáveis, são utilizadas técnicas de amostragem para reduzir o ruído e encontrar as características mais notáveis para o conjunto. A seleção de características difere nos métodos supervisionados e não-supervisionados, principalmente devido a presença do atributo classe nos métodos supervisionados. Para tarefas de agrupamento e classificação, os métodos de seleção de recursos podem ser divididos em (Hancer et al., 2020):

- **Filtros:** Esse método é aplicado para estimar o desempenho de características em relação ao conjuntos de dados. O termo filtro deriva da ideia de que atributos irrelevantes são filtrados do conjunto antes de chegarem ao algoritmo de classificação (Blum e Langley, 1997). Nas abordagens de agrupamento, todas as variáveis são avaliadas individualmente calculando a similaridade entre as amostras. Eles são geralmente mais lentos do que outras categorias de seleção de atributos, especialmente com um grande número de amostras e características.
- ***Wrappers:*** Diferentemente da abordagem de filtros, os métodos *Wrapper* são relacionados aos algoritmos de agrupamento. Essas abordagens avaliam subconjuntos de variáveis, que

permitem detectar as relações potenciais entre elas (Phuong et al., 2005). Apesar do alto custo computacional, esses métodos geralmente atingem um desempenho mais significativo quando comparados aos filtros. (Hancer et al., 2020).

- **Abordagens híbridas:** O método híbrido de seleção mescla filtros e *Wrappers*, combinando a eficiência computacional da primeira e a precisão preditiva da última abordagem (Liu e Motoda, 2007).

- ***Embedded*:** Nas abordagens embutidas geralmente são gerados rótulos para *clusters* através de algoritmos de agrupamento. Desse modo, esses métodos transformam a seleção não-supervisionada de recursos em seleção supervisionada, com aprendizado esparso a partir dos rótulos de *clusters* previamente gerados (Wang et al., 2015).

2.8 TÉCNICAS DE AGRUPAMENTO E FILOGENIA

Agrupamento é uma técnica de aprendizado não-supervisionado, utilizada para encontrar padrões inesperados em dados. A Análise de Agrupamentos é uma técnica para reunir objetos em grupos, de modo que objetos que estão no mesmo grupo são mais similares entre si do que objetos que estão em outro grupo definido, de acordo com uma medida de similaridade preestabelecida (Huang, 1997).

Uma análise criteriosa de agrupamentos necessita de métodos que proporcionem as seguintes características (Han et al., 2011):

- Escalabilidade para lidar com grandes quantidades de dados e várias dimensões;

- Capacidade para agrupar diferentes tipos de dados e objetos;

- Habilidade de definir agrupamentos de diferentes tamanhos e formas;

- Exigir o mínimo de conhecimento para determinação correta dos parâmetros de entrada;

- Executar com sucesso, mesmo na presença de ruído;

- Apresentar resultados consistentes independente da ordem em que os dados são apresentados;

Em relação aos algoritmos de agrupamento, os mais comuns podem ser divididos em grupos, como: métodos de particionamento, métodos hierárquicos, métodos baseados em densidade e métodos baseados em *grid*.

Métodos de particionamento: supondo que existam n objetos no conjunto original, métodos de particionamento dividirão o conjunto em k partições. Os algoritmos mais conhecidos desse grupo são *k-means* e *k-medoid*, os quais lidam com a ideia de centro de gravidade ou objeto central do *cluster*. Todos os métodos desse grupo possuem a mesma qualidade de agrupamento e também os mesmos problemas (Zaïane et al., 2002): é necessário conhecer o número de *clusters* antecipadamente; é difícil identificar *clusters* com grandes variações de tamanhos e o método só é adequado para *clusters* esféricos côncavos.

- **Pontos fortes**: Escalabilidade linear. Alguns desses algoritmos, como CLARANS, lidam bem com *outliers* (Swarndeep Saket e Pandya, 2016).

- **Pontos fracos**: Trabalham apenas com dados numéricos (Rokach e Maimon, 2005). São sensíveis à ordem de gravação dos dados (Park et al., 2008).

Métodos hierárquicos: um agrupamento hierárquico produz uma árvore diferenciada, frequentemente chamada de dendrograma. Nessa árvore os objetos são as folhas e os nós internos revelam a estrutura de similaridade dos pontos. Se a hierarquia de agrupamento for formada de baixo para cima, no início cada objeto de dados é um *cluster* por si só, então os pequenos *clusters* são mesclados em *clusters* maiores em cada nível da hierarquia até que na parte superior da hierarquia todos os objetos de

dados estejam em um grupo. Esse tipo de método hierárquico é chamado de aglomerativo. O processo inverso é chamado de agrupamento hierárquico divisivo. Há diversos algoritmos hierárquicos novos que apareceram nos últimos anos. A principal diferença entre todos esses algoritmos hierárquicos é como medir a semelhança entre cada par de *clusters* (Zaïane et al., 2002).

- **Pontos fortes**: Não é necessário estabelecer número inicial de *clusters*. Versatilidade e aplicabilidade com diferentes tipos de atributos (Abbas, 2008).
- **Pontos fracos**: Não são escaláveis. Há dificuldade em encontrar o número ótimo de clusters dependendo do tamanho do *dataset* (Aresu et al., 2015), (Weichselbaum et al., 2014), (Chakraborty et al., 2017b), (Korczynski, 2015)

Métodos baseados em densidade: os algoritmos baseados em densidade tentam suprir a necessidade de método hábeis de descobrir *clusters* com formatos arbitrários. Nesses métodos, a ideia de grupos baseia-se na existência de regiões densas de dados, separadas por regiões que apresentam baixa densidade. Alguns exemplos desses algoritmos são: *DBSCAN, OPTICS* e *DENCLUE*.

- **Pontos fortes**: São resistentes a ruídos e podem agrupar dados de diferentes tamanhos e formatos (Salvador e Chan, 2004).
- **Pontos fracos**: Esses métodos não lidam bem quando há grupos de densidades variáveis ou conjuntos de dados extensos. (Rathore et al., 2018) (Chakraborty et al., 2017b)

Métodos baseados em *Grid*: algoritmos baseados em *grid* primeiramente quantificam o espaço de agrupamento em um número finito de células, que forma uma estrutura em *grid* na qual todas as operações são executadas. Exemplos de algoritmos são: Sting, Wavecluster e Clique.

- **Pontos fortes**: Esses algoritmos são adequados para processamento paralelo e atualização incremental (Deng et al., 2018).
- **Pontos fracos**: O resultado do agrupamento é sensível à granularidade, isto é, ao tamanho em que os campos de dados são divididos (Xu e Tian, 2015).

Métodos baseados em fractais: esse método de agrupamento usa propriedades de autosimilaridade de dados. Primeiro, os dados são divididos em subgrupos suficientemente grandes e com alta compactação. Na sequência, o algoritmo mescla subgrupos que estão próximos uns dos outros e que possuem complexidade. Essa técnica usa um método mais natural e totalmente determinístico para gerar os *clusters* finais (Garcia e Noe, 2011). Um exemplo dos algoritmos usados nesse tipo de agrupamento é o FD3 (Sarraille e DiFalco, 2007).

- **Pontos fortes**: Eficiência e escalabilidade boas, além de lidar bem com *outliers* (Khalilian e Mustapha, 2010).
- **Pontos fracos**: Os resultados são fortemente sensíveis aos parâmetros de entrada de dados dos algoritmos (Xu e Tian, 2015).

Métodos baseados em Modelos: nesse tipo de agrupamento são utilizados modelos de referência para cada agrupamento, buscando otimizar a curva entre os elementos analisados através de um modelo matemático. Desse modo, esses algoritmos de agrupamento são ideais para modelar uma distribuição desconhecida, como uma mistura de distribuições elementares, geralmente sendo consideradas gaussianas (Fraley e Raftery, 2002). Um dos exemplos mais conhecidos é o Algoritmo de Maximização de Expectativas - EM.

- **Pontos fortes**: Essa categoria de agrupamento identifica automaticamente o número ótimo de *clusters*. Além disso, esse tipo de agrupamento considera a probabilidade de amostras pertencerem a cada grupo (Boehmke e Greenwell, 2019).

- **Pontos fracos**: Custo e tempo computacionais altos (El Attar et al., 2014).

Métodos baseados em Computação Evolutiva: O agrupamento evolutivo refere-se ao uso de algoritmos evolutivos, como algoritmos genéticos, na análise de agrupamentos. Nesse método de agrupamento são avaliadas as características que melhor representam os indivíduos, bem como os operadores de seleção, mutação e cruzamento de (Corne et al., 2010). Exemplos desses algoritmos são o KM genético (Krishna e Murty, 1999) e o algoritmo geneticamente guiado (Hall et al., 1999).

- **Pontos fortes**: Experimentos mostraram *clusters* compactos e bem separados (Nerurkar et al., 2018).
- **Pontos fracos**: O custo computacional ainda é um problema (Nopiah et al., 2010).

Métodos baseados em redes neurais: As redes neurais competitivas (RNA) são adotadas como modelos para esse agrupamento. Eles são normalmente chamados de redes auto-organizadas e são métodos, normalmente, sem supervisão. Dois algoritmos famosos são SOM e Teoria da Ressonância Adaptativa (ART)(Carpenter et al., 1991). O mapa de Kohonen (SOM) usa o conceito de vizinhança. Esse tipo de rede aprende a reconhecer seções de vizinhanças no espaço inicial, bem como a topologia dos vetores de entrada nos quais ocorre o treinamento (Haykin, 2007).

- **Pontos fortes**: Apresentam fácil capacidade de interpretação para lidar com grandes quantidades de dados e dados complexos (Du, 2010).
- **Pontos fracos**: Um bom conjunto de dados é fundamental. Além disso, determinar quais fatores são relevantes para o problema pode ser uma tarefa complicada (Barrera et al., 2010).

Métodos baseados em Distribuição: Métodos que assumem que os dados são compostos de distribuições, como distribuições gaussianas. Enquanto a distância do centro da distribuição aumenta, a probabilidade de um ponto pertencer a essa distribuição diminui. Um exemplo de algoritmo é o DBCLASD (Xu et al., 1998).

- **Pontos fortes**: Apresentam escalabilidade moderadamente alta e suporte estatístico bem desenvolvido (Xu e Tian, 2015).
- **Pontos fracos**: A complexidade de tempo é relativamente alta e tem uma forte influência de muitos parâmetros (Xu e Tian, 2015).

Métodos baseados em grafos (Teoria dos Grafos): A representação de um problema de agrupamento de dados pode usar um grafo, no qual um nó representa cada elemento do conjunto. Para modelar a distância, o método emprega um peso específico relacionado à aresta que conecta dois elementos. Exemplos de algoritmos são HCS, DTG, CLICK e CAST (Xu e Donald Wunsch, 2005).

- **Pontos fortes**: Esses algoritmos são adequados para o conjunto de dados com formato aleatório e alta dimensão (Xu e Tian, 2015).
- **Pontos fracos**: O número de *clusters* necessários deve ser predefinido (Kang et al., 2019).

Métodos baseados em *Kernel*: Esses métodos são capazes de identificar as estruturas não lineares. O agrupamento baseado em *kernel* trabalha o agrupamento implicitamente por um método *Kernel*, que realiza um mapeamento de dados de entrada linear não apropriados para um espaço de características de alta qualidade, substituindo o produto interno entre as variáveis não-lineares por um *kernel* adequadamente determinado (Shawe-Taylor e Cristianini, 2004). Como exemplos citam-se K-Means Kernel e Agrupamento com suporte vetorial(SVC).

- **Pontos fortes**: Esses métodos são capazes de identificar estruturas não-lineares (Chitta et al., 2011).

- **Pontos fracos**: Algoritmos são complexos e a complexidade de tempo é alta (Chitta et al., 2011).

Métodos baseados em *Fuzzy*: esse tipo de agrupamento permite que um indivíduo pertença parcialmente a mais de um grupo, com graus variados de vizinhança. Elementos de contorno de grupo não precisam pertencer inteiramente a um cluster, porém graus de associação entre 0 e 1 são atribuídos, indicando sua associação parcial (Bora et al., 2014). Geralmente, esses modelos são frequentes no reconhecimento de padrões. Os algoritmos Fuzzy c-means e Fuzzy c-shells são exemplos de agrupamento do tipo *fuzzy*.

- **Pontos fortes**: Esses métodos apresentam robustez para ambiguidade e mantêm mais informações do que qualquer método de agrupamento rígido (Askari, 2021).

- **Pontos fracos**: Possuem alta sensibilidade a ruídos (Gosain e Dahiya, 2016).

Ressalta-se que a tarefa de agrupamento de elementos é altamente dependente dos parâmetros, medidas de similaridade e métodos usados pelo algoritmo escolhido(Metz, 2006). A seguir serão abordados algoritmos utilizados dentro dos métodos hierárquicos, os quais serão utilizados no presente trabalho.

2.8.1 Técnicas de Agrupamentos de Classificação Hierárquica

Algoritmos hierárquicos são técnicas de agrupamentos que criam uma hierarquia de relacionamentos entre os objetos. Eles funcionam de duas formas: na aglomerativa opera-se criando agrupamentos a partir de elementos isolados. Já na divisiva, inicia-se com um grande conjunto, o qual vai sendo quebrado em partes até chegar a elementos isolados (Linden, 2009).

Alguns dos principais métodos de agrupamento, os quais serão utilizados neste trabalho, são listados a seguir:

- ***Average Linkage* ou Ligação Média**: o método de ligação média também é chamado de método de grupos de pares ponderados. Esse algoritmo define a distância entre os grupos como a distância média entre cada um dos membros, ponderada de modo que os dois grupos tenham igual influência no resultado.

- ***Complete Linkage* ou Ligação Completa**: na ligação completa, também conhecido como ligação pelo vizinho mais distante ou método máximo, define-se a distância entre dois conjuntos como a distância entre os dois membros mais distantes. Esse método geralmente gera *clusters* que são bem separados e compactos.

- ***Single Linkage* ou Ligação Simples**: o método de ligação simples, também conhecido como agrupamento dos vizinhos mais próximos, é um dos mais antigos e famosos dos agrupamentos hierárquicos. A distância entre dois grupos é definida como a distância entre os dois elementos mais próximos. Isso geralmente produz *clusters* em que os indivíduos são adicionados sequencialmente a um único grupo.

- **Método de Ward**: no método de Ward, os grupos são formados de modo que a soma dos quadrados agrupados dentro dos grupos seja minimizada. Isto é, em cada divisão, os dois *clusters* gerados são fundidos. Isso resulta em menor aumento na soma dos quadrados agrupados dentro dos grupos.

Após a decisão pelo método mais apropriado e sua aplicação, são gerados agrupamentos ou *clusters*. Ressalta-se que a interpretação dos *clusters* gerados é uma tarefa complexa, que objetiva descobrir significados relacionados à área dos conjuntos de dados analisados. Outros assuntos relacionados aos métodos de agrupamento, como medidas de similaridade e também suas árvores resultantes serão abordadas nas próximas subseções.

2.8.2 Métricas de Similaridade e Dissimilaridade

As medidas de similaridade são medidas numéricas para demonstrar quão parecidos dois objetos são. Quanto mais similares, maior é a medida de similaridade resultante. Geralmente utilizam escalas de 0 a 1, no qual quanto mais próximo de 1, os elementos são menos similares. Enquanto a dissimilaridade é uma medida numérica do quão diferente dois objetos são. Desse modo, objetos similares possuem dissimilaridade baixa. Os índices de distância são tipos especiais de dissimilaridade.

Como exemplos de índices de distâncias podem ser mencionadas a distância euclidiana, Manhattan e a distância de Minkowski (Tan et al., 2006). Já as medidas de proximidade são aplicadas sob vetores binários, como por exemplo, as medidas Jaccard e *Cosine*. Na Tabela 2.6 são apresentadas descrições e fórmulas de outras medidas conhecidas na literatura.

Tabela 2.6: Principais medidas de similaridade encontradas na literatura de análise de *malware*

Nome	**Descrição**	**Equações**
Distância Manhattan	Essa medida calcula a distância entre dois pontos medidos ao longo de eixos em ângulos retos (Sanz et al., 2014).	$MD(i,j) = \sum_{i=1}^{n} \lvert i_x - j_x \rvert$, no qual i e j = o primeiro e o segundo ponto, respectivamente. I $_x$ e j $_x$ são componentes dos pontos i e j, respectivamente
Distância Euclidiana	Essa medida é o comprimento do segmento de linha que liga dois pontos (Sanz et al., 2014).	$ED(x,y) = \sum_{i=1}^{n} \sqrt{v_i - u_i}$, no qual u$_i$ e v$_i$ são os componentes dos pontos i e j, respectivamente.
Semelhança por Cosseno (Cosine)	Essa semelhança emprega o cosseno do ângulo entre dois vetores (v e u) do espaço de características dos pontos x e y, respectivamente (Sanz et al., 2014).	$Cosine(x,y) = \frac{v.u}{\lvert\lvert v \rvert\rvert.\lvert\lvert u \rvert\rvert}$
Índice de Jaccard	Essa medida estima a similaridade entre os conjuntos de amostras a e b, usando o tamanho da interseção dividido pelo tamanho da união dos conjuntos de amostras a e b (Pandit e Gupta, 2011).	$Jaccard(a,b) = \frac{\lvert a \cap b \rvert}{\lvert a \cup b \rvert}$
Distância por Compressão Normalizada (NCD)	NCD é uma medida baseada na aplicação de compressores para calcular a razão de similaridade de dois arquivos (Cebrián et al., 2007).	$NCD(p,q) = \frac{C(pq) - min\{C(p).C(q)\}}{max\{C(p).C(q)\}}$
Jaro Winkler	Adaptação da métrica de Jaro (Jaro, 1995). Essa medida emprega o número e a ordem dos caracteres familiares entre duas *strings* para calcular a similaridade (Winkler, 1999).	$J = \frac{m}{3a} + \frac{m}{3b} + \frac{m-t}{3m}$, no qual m é o número de correlações entre os caracteres; a e b são os respectivos tamanhos das strings comparadas; t é o número de transposições.

Coeficiente de correlação de Pearson	esse coeficiente mede a relação estatística (associação) entre duas variáveis (Katos, 2007).	$CCP = \frac{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}}$

2.8.3 Dendrogramas

Um conjunto de dados, normalmente, é composto por diversos grupos e esse grupos, por sua vez, são formados de subgrupos e assim continuamente. Essa sucessão de níveis e subníveis constitui naturalmente uma hierarquia (Hart et al., 2000). Para demonstrar a sequência com que os agrupamentos foram formados, graus de similaridade e distribuição dos *clusters* é interessante a utilização de dendrogramas.

Dendrogramas são árvores nas quais os exemplares são apresentados no eixo horizontal, enquanto a similaridade com que os grupos são formados é indicada no eixo vertical. A estrutura da árvore tem N folhas e altura N-1, sendo que o primeiro nível corresponde a um agrupamento contendo todos os N exemplares do conjunto de dados (Metz, 2006). Um exemplo de dendrograma é apresentado na Figura 2.6.

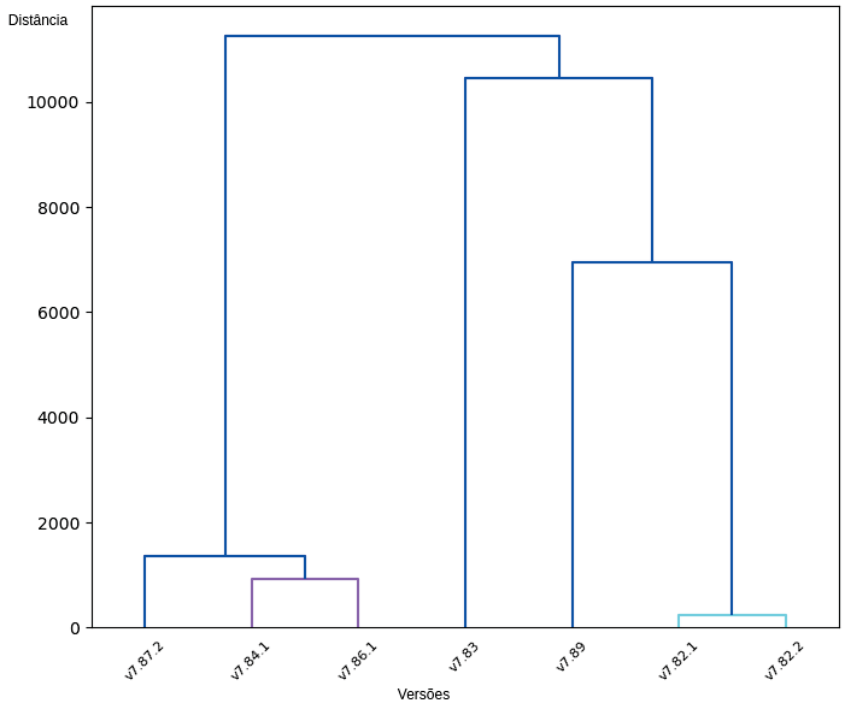


Figura 2.6: Dendrograma de versões do aplicativo Netflix.

É importante ressaltar que a escolha da medida de similaridade ou dissimilaridade e do algoritmo de agrupamento influencia no dendrograma gerado. Portanto, a análise visual de um dendrograma utilizando distância de Manhattan e Ligação Completa deve ser diferente da análise de um dendrograma gerado a partir de Ligação Média e da distância euclidiana (Metz, 2006). Uma maneira de avaliar o grau de deformação da construção do dendrograma é a partir do coeficiente de correlação cofenético, proposto por Sokal e Rohlf (1962).

Desse modo, para escolha das técnicas recomenda-se o seguinte método (Vicini, 2005):

- Formulação do problema
- Tratamento/padronização dos dados

- Escolha de um coeficiente de semelhança
- Escolha de um algoritmo de agrupamento
- Avaliação e interpretação dos resultados

A decisão sobre o agrupamento final também é chamada corte do dendrograma. O corte do dendrograma é similar ao desenho de uma linha por todo o dendrograma para especificar o agrupamento final. Embora não exista um padrão definitivo para o corte ótimo, existem alguns critérios indicados na literatura (Bruzzese e Vistocco, 2010).

Após essa introdução sobre os dendrogramas, a próxima subseção abordará a Filogenia e sua estrutura hierárquica.

2.8.4 Validação de agrupamentos

Em atividades de agrupamento, diferentemente de algoritmos de aprendizado supervisionado, a rotulagem completa de dados de referência (*ground truth*) geralmente não está disponível para o conjunto de dados analisado. Desse modo, a validação de resultados pode ser complexa, geralmente envolvendo análises manuais feitas por especialistas (Perdisci e U, 2012).

Todavia, ao longo das últimas décadas diversas formas de validação foram propostas . As principais abordagens de índices de validação podem ser divididas em três grupos: critérios externos, critérios internos e critérios relativos (Halkidi et al., 2001).

- **Critérios externos:** A ideia básica dos índices baseados em critérios externos é testar se a estrutura dos *clusters* é aleatória ou não. O objetivo principal desses índices é medir o valor de compatibilidade entre uma hierarquia ou partição obtida e uma partição de referência dos mesmos dados. Entre os mais conhecidos estão: Rand Index, Jaccard Coefficient, Rand Index Ajustado e Fowlkes-Mallows.
- **Critérios internos:** Índices de validação baseados em critérios internos utilizam características e quantidades inerentes ao próprio conjunto de dados.
- **Critérios relativos:** Avalia qual dentre duas ou mais estruturas de grupos é melhor sob algum aspecto. Geralmente são critérios internos capazes de quantificar a qualidade relativa.

Um dos casos em que é recomendado o uso de índices de critérios internos é na validação de agrupamentos hierárquicos, sendo sugerido o Coeficiente de Correlação Cofenética. Esse coeficiente mede o nível de preservação das distâncias dos objetos no dendrograma em relação às distâncias originais (Ferreira, 2008). O Coeficiente de correlação cofenética (CCC) é apresentado a seguir:

$$CCC = \frac{Cov(F, C)}{\sqrt{V(F)V(C)}}$$

Em que: F representa a matriz fenética; C retrata a matriz cofenética.

2.8.5 Filogenia

Filogenia é a história genealógica de um grupo de organismos e uma representação hipotética das relações ancestral/descendente (Hennig, 1966). O relacionamento entre as espécies pode ser representado por árvores binárias (com ou sem raiz) ou por uma rede (Liu et al., 2017). Para árvores com raízes, a raiz representa o ancestral comum de todas as espécies representadas na árvore. Quando um ancestral não é especificado ou presumido, uma árvore sem raiz é útil para representar os relacionamentos entre os grupos, independente do ancestral comum. Os nós na filogenia representam a unidade da taxinomia ou a espécie.

A filogenia difere de taxonomia porque essa é baseada em um grupo com características compartilhadas, enquanto aquela assume um ancestral comum e busca derivar os relacionamentos entre os seus descendentes (Khoo e P.Lió, 2011).

A filogenética computacional é a aplicação de algoritmos, métodos e programas computacionais para análises filogenéticas. A pesquisa de filogenia de *malware* inspira-se na filogenética da Biologia. Métodos comumente utilizados na biologia para construir, avaliar e comparar árvores filogenéticas podem ser adaptados para pesquisa de programas maliciosos (Karim et al., 2005a).

Os métodos mais comuns para construção de árvores são baseados em distância, parcimônia, máxima verossimilhança e abordagens bayesianas. Por exemplo, um dos métodos mais comuns de construção de árvores filogenéticas é o MP – *Maximum Parsimony Method* ou princípio da máxima parcimônia (Ye et al., 1983). Esse método assume que a árvore mais precisa é aquela que requer menos mudanças para produzir os dados contidos no alinhamento (Kim e Warnow, 1999). Outro método popular é a estimativa por máxima verossimilhança (*maximum-likelihood estimation* - MLE) utilizado para estimar os parâmetros de um modelo estatístico.

Os métodos MP e MLE são baseados em caracteres, ou seja, dependem dos valores individuais de cada sequência de alinhamento. Existem outros métodos conhecidos como métodos baseados em distância, os quais são o foco desta pesquisa. Na filogenia de estudos de biologia, é considerada uma sequência inteira de taxonomia, frequentemente utilizando uma matriz de distância. O método mais frequentemente encontrado nessa literatura é o método de ligação média.

Ressalta-se que tanto os métodos baseados em caracteres quanto os métodos baseados em distância possuem vantagens e desvantagens. O método UPGMA, por exemplo, é de fácil implementação, porém não é muito utilizado para criação de árvores filogenéticas por não levar em consideração diferentes taxas de mudanças evolutivas entre sequências representadas na árvore (Haubold e Wiehe, 2006). Métodos baseados em caracteres são mais fiéis na representação evolutiva da filogenia, mas são mais caros computacionalmente para implementar.

2.8.6 Agrupamento em Filogenia de *Malware*

Embora métodos filogenéticos que estimam a máxima verossimilhança são populares na Biologia, no domínio de *malware* existem diversas limitações. Por exemplo, dada uma múltipla sequência de alinhamento e uma árvore filogenética específica, a verossimilhança para cada caractere no alinhamento deve ser calculada. Com n sequências e s estados, a complexidade computacional do cálculo será $O(s^{n-2})$ (Huson et al., 2010). Isso já é considerado caro quando consideram-se dados como o DNA no qual existem apenas 4 estados (A, G, T, C), mas em relação a *malware* nos quais as instruções assembly são os estados, o número de s está na casa das centenas (Anderson et al., 2011).

Outra limitação relacionada à análise e filogenia de *malware* é a compressão das cópias realizadas da memória da máquina virtual durante procedimentos de análise dinâmica. James Fowler (Fowler, 2016) apresentou resultados significativos com amostras de *malware*, com o projeto VirusShare. Esse projeto precisaria do total de 9 Petabytes de memória RAM em uma máquina virtual atual para ser analisado dinamicamente .

Pfeffer et al. (2012) apontam como alternativa o uso de outras formas de análise para extração de características dos *malware*, como a genética e a baseada em linguística.

Dumitras e Neamtiu (2011) apontaram dois grandes desafios na determinação da linhagem e na proveniência de *malware*. O primeiro está no agrupamento de amostras em famílias. As abordagens atuais usam características como assinaturas ou grafos de chamadas para identificar variantes da mesma família. A falta de capacidade das ferramentas atuais para descompactar códigos ofuscados ou gerar grafos de chamadas dinâmicas, torna difícil expor a conexão entre variáveis independentes. O outro desafio apontado pelos autores envolve a montagem da árvore filogenética. A falta de informações relevantes dos criadores de *malware*, como processos de desenvolvimento do código e estratégias de disseminação, impede os analistas de estabelecer uma verdade fundamental para a validação dos resultados da linhagem gerada. Walenstein e Lakhotia abordam, no estudo "*A Transformation-based Model of Malware Derivation*" (Walenstein e Lakhotia, 2012a), as principais implicações atuais para a construção dos relacionamentos (evolutivos ou

não) dos *malware*. Segundo os autores, dois problemas atuais importantes na captura de evolução em relação ao *malware* são a falta de clareza conceitual e a falta de um formalismo apropriado e necessário para modelar os relacionamentos (Walenstein e Lakhotia, 2012b). Outros problemas abordados, por exemplo, são a descendência de múltiplas linhagens, o compartilhamento de atualizações de funcionalidades de múltiplas variantes e diferenças nos mecanismos de geração e compilação de código.

3 REVISÃO DA LITERATURA

No presente capítulo são mencionados e discutidos trabalhos com diversos estilos de classificação de códigos maliciosos em famílias.

Foram revisados vários estudos que envolveram filogenia como técnicas de agrupamento de variantes de *malware* Android. O método aplicado utilizou como base o paradigma GQM (*Goal-Question-Metrics*) (Caldiera e Rombach, 1994), conforme apresenta a Tabela 3.1.

Tabela 3.1: Objetivo do Mapeamento Sistemático segundo o paradigma GQM.

Analisar	Publicações científicas através de um estudo baseado em mapeamento sistemático
Com o propósito de	Identificar abordagens, técnicas, medidas, configurações de experimentos
Em relação a	Classificação de variantes de *malware* Android em famílias utilizando filogenias com métodos de agrupamento
Do ponto de vista dos	Pesquisadores
No contexto de	Análise de *malware*

A etapa seguinte é definir as questões de pesquisa (QPs). A QP determina o que deve ser respondido no final do mapeamento. Primeiramente, foi definida a seguinte questão principal :

QP : Quais os parâmetros, como medidas de similaridade/dissimilaridade e algoritmos de agrupamento, são utilizadas em filogenias na classificação de variantes de *malware* Android em famílias?

Além disso, outras RQs foram definidas para alcançar o objetivo principal do MSL.

QP1: O estudo aborda relacionamentos hierárquicos ou de linhagem entre as variantes e famílias?

QP2: Qual o tipo de agrupamento aplicado?

QP3 : Qual a medida de similaridade utilizada?

QP4: Foi utilizado algum método supervisionado em conjunto ao algoritmo não-supervisionado?

3.0.0.1 Estratégia de Pesquisa :

A próxima etapa da metodologia é a definição da estratégia de pesquisa. Nessa etapa, as palavras-chave de pesquisa são elaboradas. Foram escolhidas as bases ACM, IEEE e Springer. A seguinte palavra-chave foi utilizada na busca "Android Malware"AND "Malware Phylogeny"AND ("Family Classification"OR "Families Classification") AND "Clustering"AND "Similarity Measure"OR "Malware Lineage"OR "Malware Function Similarity Techniques". Em cada base de dados a palavra de pesquisa teve alterações sintáticas de acordo com os padrões da base. Foi selecionada a área *"Computer Science"* nas bases que ofereciam opções de áreas e disciplinas. Restrições como tipo, localização e ano de publicação não foram aplicadas. A Tabela 1 apresenta o número de resultados de pesquisa por base de dados.

3.0.0.2 Seleção de estudos :

A seleção dos estudos foi realizada depois das buscas, sendo dividida em duas fases. Na primeira etapa, os títulos, palavras-chave e resumos foram lidos. Após isso, os estudos foram selecionados de

acordo com o critério de inclusão (IC). A segunda fase excluiu *papers* selecionados na primeira fase de acordo com o critério de exclusão (EC).

Foi utilizado um critério de inclusão:

IC1: *papers* aplicando métodos de agrupamento para classificação de *malware* Android em famílias.

E quatro para exclusão de estudos:

EC1: *papers* que não estavam completamente disponíveis;

EC2: *papers* que apareciam em mais de uma base;

EC3: *papers* que não estavam em inglês; e

EC4: livros

Embora a palavra de pesquisa definida seja específica para classificação de *malware* Android, foram encontrados falsos positivos. Como as palavras-chave continham "Android *malware*", alguns trabalhos relacionados a detecção de *malware* Android (classificação entre aplicação maliciosa ou não-maliciosa), sendo que o foco da pesquisa é o de classificação em famílias (multi-classe). Desse modo, o IC1 foi importante para a seleção de artigos.

O critério de inclusão incluiu 173 dos 556 encontrados nas bases de pesquisa. A aplicação dos critérios EC1 e EC4 eliminou 97 trabalhos. Foram selecionados 76 nesse processo.

Além disso, foi realizada pesquisa no Google Acadêmico e mais 6 artigos foram incluídos. As datas de publicações dos estudos ocorreram entre 2011 e 2022.

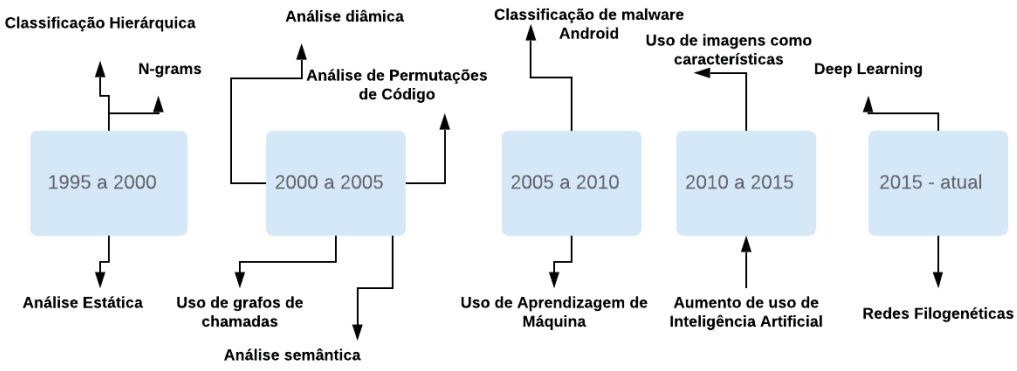
3.0.0.3 Extração de dados:

A fase da extração de dados é a ultima desse mapeamento sistemático da literatura. Os 82 artigos selecionados na última etapa foram lidos integralmente para extração das informações.

Devido à grande diversidade de métodos na literatura, como medidas de similaridade/dissimilaridade, ferramentas de análise, tipos de representação de conjuntos de dados, os artigos foram divididos por anos, entre as Tabelas 3.3 a 3.9. Informações adicionais estão disponíveis nos gráficos das Figuras 3.2 a 3.5.

Ao longo da pesquisa bibliográfica, verificou-se que de acordo com a evolução da pesquisa científica e das técnicas de análise, os métodos aplicados para investigação de famílias de *malware* também apresentaram mudanças. Partindo disso, na Figura 3.1 apresenta-se uma linha do tempo com as principais características encontradas nos trabalhos de agrupamento e filogenia de *malware* em geral.

Figura 3.1: Linha do tempo e características de pesquisas em classificação e filogenia de *malware*



A seguir são abordados trabalhos agrupados por semelhanças, como objetivos ou métodos utilizados.

3.1 ANÁLISE FILOGENÉTICA DE *MALWARE*

Em 1998, Goldberg et al. (1998) utilizaram sequências de 20 bytes e métodos de coincidência booleana para comparação de exemplares. Após a geração de vários modelos de filogenias de *malware*, os autores provaram que sequências de 20 bytes são grandes o suficiente para inferir que um espécime é derivado de outro. A filogenia perfeita é abordada nesse trabalho, ou seja, um grafo onde todos os exemplares que compartilham um estado necessitam estar conectados.

Em 2004, Ero Carrera e Gergely apresentaram uma nova ferramenta de filogenia para análise de similaridade de *malware*, baseada em grafos heurísticos, desenvolvida em Python e com *IDA - Interactive Disassembler Pro* (Carrera e Erdélyi, 2004). Os exemplares de *malware* foram comparados por comportamento através de funções extraídas estaticamente como recursos. Foram estudadas seis famílias de *malware* nesse estudo e os autores relatam, como limitações do trabalho, a compactação e criptografia do *malware* ser comum, sendo uma inconveniência para a geração totalmente automática dos grafos e da abordagem de geração, comparação e classificação dos binários. Sugerem como sugestão de trabalho futuro a criação de um banco de dados de grafos, para armazenamento das representações e propriedades dos binários.

Em 2005, os mesmos autores publicaram o estudo "*Malware Phylogeny Using Maximal* Π *Patterns*"(Karim et al., 2005b). O método criado objetivou encontrar modificações e permutações em sequências de proteínas, introduzido por Eres et al. (2003). Para o experimento com Π *Patterns* em filogenias de *malware*, foram utilizadas 18 amostras de softwares maliciosos para sistema operacional Windows de uma base de dados on-line (vx.netlux). As 18 amostras geraram 1570 "Π *patterns*" máximos. Para comparação de similaridade foram utilizadas as técnicas XNOR e *Cosine*. Uma limitação do estudo apresentado, apontada pelos autores, é a determinação do exato valor do limite em relação a Π *patterns* para decidir se um *malware* é uma variante de algum já conhecido ou se foi desenvolvido de forma independente.

O BitShred, desenvolvido por Jiyong Jang, David Brumley, Shobha Venkataraman em 2010, é um conjunto de ferramentas para triagem de *malware* em larga escala (Jang et al., 2010). O software utiliza técnicas de mineração de dados, estrutura de dados probabilísticos em conjunto com funções *hashing* para análise de *malware*. Uma inovação dessa abordagem é o uso eficiente da memória, resultando em economia em 82 vezes, em comparação com tentativas mais antigas. Uma das formas de visualizar as famílias identificadas pelo BitShred é mediante árvores filogenéticas. De acordo com os autores, quanto mais parecidos os exemplares são, mais próximos se encontram na árvore. Os resultados das famílias encontradas pelo algoritmo filogenético foram confrontados e confirmados com análises manuais.

Khoo e P.Lió (2011) utilizaram duas famílias de *malware* para estudo de árvores e redes de filogenia a partir de engenharia reversa. Foram analisados logos, mapeamento de chamadas de API e padrões de sequencias de *strings* para definição do *framework* proposto. Ao todo foram realizados 6 experimentos com as famílias FakeAV, Skyhoo e um conjunto de softwares benignos (*goodware*). Os autores acreditam que devido ao número baixo de amostras de goodware, menos de 25 arquivos, o número de falsos positivos foi considerado alto.

Guan et al. (2012) utilizaram análise homóloga para gerar filogenia de *malware*. Foram utilizadas técnicas como Modelo Oculto de Markov (HMM) e ferramentas de bioinformática para extração das características. Nesse trabalho, as sequências existentes nos *malware* foram comparadas a sequências de proteínas, transformando cada letra em dois aminoácidos. As sequências de aminoácidos geradas foram comparadas utilizando a ferramenta UGENE. Apesar dos experimentos serem realizados com uma amostra pequena (46 exemplares), os autores afirmam que o algoritmo foi capaz de distinguir as três diferentes famílias analisadas.

Ravi et al. (2013) disponibilizaram um método para análise de *malware* baseado em comportamento usando PHMM – *Profile Hidden Markov Models*. Assim como a filogenia vem da biologia, o PHMM é um esquema desenvolvido especialmente para modelagem de similaridade entre sequencias biológicas como proteínas e DNA. No esquema criado pelos autores são utilizados arquivos XML gerados a partir da análise dinâmica dos programas maliciosos investigados. Para cada família de *malware* são

usados de 5 a 20 arquivos em média. Apesar dos bons resultados iniciais, os autores ressaltam que o método precisa ser aplicado em larga escala e também para *malware* nunca vistos.

Modelos baseados em árvores são os principais no estudo da filogenia de *malware*. Entretanto, não são os mais adequados para representar eventos de reticulação que, ocasionalmente, ocorrem na geração de novas variantes. Liu et al. (2016) utilizaram redes *Median-Joining* (MJ) baseadas em sequências de chamadas a APIs para construir redes filogenéticas. Para os experimentos utilizou-se uma base com 3357 amostras de várias famílias. Os resultados apresentados demonstram que o método proposto é útil para revelar relacionamentos existentes com as famílias dos *malware* estudados. Na continuação da pesquisa, Liu et al. (2017) utilizaram grafos para inferir redes de filogenia. Foi utilizado um algoritmo húngaro para cálculo da distância edit dos grafos de cada par de amostras. A distância euclidiana foi aplicada sob a frequência de instruções mnemônicas extraídas. Grafos de filogenias verdadeiras de 5 famílias foram criados, das quais 3 famílias são de *malware*. A maior taxa de acurácia atingida nos experimentos foi de 64,8%.

Gutiérrez-Cárdenas e Orihuela (2016) apresentaram resultados parciais de suas pesquisas com Filogenia de *malware*. Para a pesquisa foram utilizadas poucas famílias de *malware* para o sistema operacional Windows. Foram analisadas as chamadas de sistemas e bibliotecas utilizadas, as quais foram extraídas estaticamente e comparadas mediante HexEdit . Para a construção da árvore filogenética foi utilizado o algoritmo *Neighbour-Joining*. Apesar dos autores não apresentarem dados sobre eficiência do algoritmo ou comparação com softwares benignos ou com antivírus, pretendem continuar as pesquisas aumentando o número de famílias e de amostras de *malware*.

Oyen et al. (2017) apresentaram um algoritmo Bayesiano de descoberta de redes para aprendizado de DAG – Grafo Acíclico Direto , utilizando inferência estatística de dependências condicionais a partir de dados observados pré-informativos sobre a ordenação parcial de variáveis. O algoritmo foi aplicado para aprender grafos filogenéticos de três famílias de *malware* e duas famílias benignas. Foram utilizadas cadeias de Markov para representar as transições sequenciais dos programas. Os resultados demonstraram que o conhecimento prévio permitiu melhorias na acurácia dos relacionamentos dos programas e seus ancestrais.

3.2 IDENTIFICAÇÃO DE VARIANTES

O trabalho de Georg Wicherski introduz uma forma não-criptográfica, rápida de calcular uma função *hash* para binários no formato portável (*Portable Format*) (Wicherski, 2009). O algoritmo proposto, peHash, transforma informações estruturais sobre uma amostra de *malware* em um valor de *hash*. Os testes realizados em uma base de exemplares da Alliance mostraram que o algoritmo ajudou a revelar relações entre novas amostras e ameaças já conhecidas. Todavia, uma limitação existente e comentada pelo autor é a possibilidade de colisão de *hashes*.

Ye et al. (2010a) desenvolveram uma abordagem híbrida para agrupamento hierárquico de *malware*, utilizando técnicas de inteligência artificial. Para representação das características dos programas maliciosos foram utilizadas as frequências das instruções de funções baseadas em sequencias de instruções. Na abordagem híbrida de clusterização, combinou-se agrupamento hierárquico com algoritmo K-medoids. Uma inovação encontrada neste trabalho é o uso de agrupamento Ensemble. Esse termo refere-se a obter um consenso e uma solução ótima de um número de entradas de um conjunto de dados em particular. Apesar da desvantagem do uso da análise estática, devido à rapidez de sua classificação (30 mil amostras em 20 minutos), o sistema foi incorporado em um antivírus comercial (Kingsoft). O sistema demonstrou taxas de detecção e classificação de *malware* superiores a de cinco softwares de comerciais.

Ye et al. (2010b) focaram na análise de arquivos da denominada lista cinza: arquivos que não são classificados nem como maliciosos nem como benignos. Os arquivos contidos nesse tipo de grupo ou são autenticados ou são rejeitados pelos analistas. Entretanto, os autores afirmam que devido ao uso de novas técnicas de criação de *malware*, a lista cinza tem crescido constantemente. O classificador proposto, HAC, combina dois níveis de precisão, utilizando APIs do Windows chamadas pelos arquivos portáteis (*PE Files*). O primeiro nível são utilizadas regras de alta precisão de classificação de arquivos benignos

e regras de baixa precisão de arquivos considerados maliciosos. No segundo nível, classificam-se os arquivos das classes minoritárias e otimiza-se a precisão, mediante pós-processamento, como a "*Best Fit Rule*". O classificador desenvolvido foi testado em uma base de mais de 8 milhões de *malware*, sendo posteriormente incorporado ao antivírus da Kingsoft.

Nakazato et al. (2011) apresentaram um método de agrupamento de softwares maliciosos utilizando frequências de funções a partir de traços de execução, utilizando *threads*. Para o cálculo da frequência foi utilizado o método do *N-gram* e TF-IDF. As características foram extraídas a partir de traços de execuções de chamadas a APIs do Windows. Na classificação média de 2.312 amostras foi obtida precisão de 55% e *recall* de 90%. Os autores ressaltam que a acurácia do agrupamento é influenciada pelo algoritmo de corte da árvore gerada. Uma contribuição desse trabalho é a determinação automática do número de agrupamentos, sem processo de pré-aprendizagem das famílias de *malware*. Os autores encontraram 39 amostras maliciosas com características diferentes de outros exemplares, sugerindo que tratavam-se de novos tipos de *malware*.

Kinable e Kostakis (2011) estudaram agrupamento de variantes de *malware* a partir de grafos de chamadas. Algoritmos como *k-medoids* e *DBSCAN* foram utilizados para determinar a similaridade de binários a partir dos grafos gerados, relatando que o último apresentou melhor performance no conjunto de amostras investigadas. Apesar das métricas aplicadas , foi mencionada pelos autores a dificuldade para determinação do número de *clusters* para o algoritmo *k-medoids*. Concluiu-se que esse algoritmo não foi eficaz na determinação das famílias dos *malware* analisados. Ressalta-se ainda que os *clusters* gerados pelos algoritmos foi validado por analistas.

Anderson et al. (2011) apresentaram um algoritmo de detecção de variantes de *malware* baseado em grafos usando análise dinâmica. Os grafos representavam cadeias de Markov, onde os vértices são instruções e as probabilidades de transição são estimadas pelos dados contidos nos traços. Foi utilizada uma SVM para executar a classificação a partir da matriz de similaridade criada com base nos grafos resultantes. Alguns anos mais tarde, em 2016, o trabalho foi continuado em Oyen et al. (2017). Os autores continuaram utilizando grafos e análise híbrida, assim como a ideia de múltiplas visões para descrever os dados extraídos dos *malware*. No novo trabalho foram analisados traços de instruções dinâmicas, chamadas do sistema e, da análise estática, foi utilizado o grafo de fluxo de controle (CFG) do binário desmontado. Outra inovação foi a adoção de redes Bayesianas para combinar conhecimentos de especialistas humanos com dados estatísticos.

Alzarooni (2012), em sua tese de doutorado, propôs um novo método para detecção de variantes *malware* utilizando análise semântica. A metodologia proposta é realizada em 3 partes: criação da assinatura, análise de fatiamento e teste de geração de dados. O autor afirma que a principal complexidade na criação de uma assinatura semântica para o *malware* é conseguir o equilíbrio entre otimizar a taxa de detecção e ter um desempenho adequado.

Arun Lakhotia e outros autores publicaram, em 2013, o sistema VILO (Lakhotia et al., 2013). Utilizando Aprendizagem de Máquina, o sistema desenvolvido tinha como objetivo realizar classificação e triagem de *malware* rapidamente. Foram extraídas estaticamente características mediante técnicas utilizadas em trabalhos anteriores dos autores, como "N-perms". Os resultados demonstraram que com apenas treinamento de menos de 20 amostras por família já era o suficiente. Apesar de o nome do sistema ser relacionado à Filogenia (VI – Vírus, Lo -fiLOgenia), os autores não apresentam estudos do relacionamento evolutivo das espécies e famílias classificadas pelo software. Não é gerada uma árvore filogenética também.

Islam et al. (2013) desenvolveram, segundo os autores, o primeiro método integrador de características dinâmicas e estáticas, a partir de um único teste. Na extração das características foram utilizadas frequências dos comprimentos das funções , informações de *strings* imprimíveis e chamadas a APIs. Entre as técnicas de aprendizagem de máquina utilizadas, destaca-se o desempenho da Floresta Randômica (*Random Forest*). Além disso, os autores consideram que a data aproximada de liberação do *malware* impactou nos resultados dos testes realizados, apresentando melhor desempenho para códigos mais antigos quando comparados a novas variantes detectadas.

Edem et al. (2014) investigaram o uso de K-means, no agrupamento de variantes polimórficas de *malware*, no suporte à investigação forense. A partir da análise dinâmica, comportamento dos binários foram convertidos em vetores e foi aplicada função *hash* usando LSH (*Hashing* Sensitivo à Localidade). Destaca-se que foram extraídas características do registro e de características de arquivos DLLs. A ferramenta desenvolvida apresenta diversas funcionalidades, inclusive armazenamento em banco de dados, geração de relatórios em XML e funções para *backup*. Os autores ressaltam a necessidade de ferramentas que integrem análise estática e dinâmica para os investigadores forenses. Entretanto, relembram que a taxa de acurácia e desempenho computacional precisam ser considerados.

Annachhatre et al. (2014) avaliou o algoritmo *k-means* para agrupamento de *malware* , em sua dissertação de mestrado. Foram utilizados número de grupos iniciais (k) de 2 a 15 nos agrupamentos de classificação em famílias, sendo que k=9 foi o melhor resultado nos experimentos com 9 mil amostras. O modelo oculto de Markov foi avaliado com diversos geradores de *malware* e compiladores. Para trabalhos futuros, a autora cita o uso de outras medidas de distância e refinamento de agrupamentos com famílias dominantes.

Kim et al. (2014) abordaram a relação da velocidade e acurácia em detecção de variantes de *malware*. Os autores desenvolveram duas ferramentas: a I-Filtering e a I-Filter. A primeira foi apresentada no trabalho anterior, sendo que seu desempenho foi comparado com a I-Filter. A técnica utilizada foi baseada em comparação de valores *hashes*, equivalente à uma comparação de correspondências. Cita-se o uso de SCFS (controle estruturado do fluxo de *strings*) e da distância Edit na análise de similaridade das amostras. A evolução da nova ferramenta comprovou que o desempenho melhorou 1043 vezes quando comparada à I-Filtering. No ano seguinte, duas novas ferramentas são apresentadas: Malfinder e Malcore. Kim et al. (2015) focaram no gargalo de desempenho existente no sistema SCFS: quanto maior a acurácia, menor o desempenho computacional e maior o tempo gasto. Em comparação à ferramenta anterior, Malcore teve uma otimização de desempenho em 289 vezes. Apesar da melhora no desempenho computacional com o Malfinder, as taxas de detecção de variantes das famílias estudadas ficaram entre 43% e 66% Kim e Park (2015).

Narra et al. (2015) compararam o uso de SVM (máquina de vetor de suporte) e algoritmos de agrupamento para detecção de *malware*. Como vantagem do uso de agrupamento citam que novas famílias podem ser classificadas antes dos modelos serem treinados para uma família específica. Nos experimentos com agrupamento, foram comparados os algoritmos *K-means* e o algoritmo de Maximização de Expectativas (EM), a partir de modelos ocultos de Markov. Como resultado, *K-means* e SVM obtiveram índices semelhantes no agrupamento de *malware*, enquanto o método EM obteve resultados inferiores. Todavia, os algoritmos de agrupamentos não precisavam de indicadores de software benigno ou maligno, bastando informações de famílias de *malware* preexistentes.

Gharacheh et al. (2016) também utilizaram Modelos Ocultos de Markov (HMM) para detecção de variantes de *malware*. Porém, apresentam um modelo de duas camadas visando superar os desafios advindos de softwares maliciosos metamórficos. Como características são utilizados *opcodes*, que foram extraídos com eliminação de sequências parciais. Essas sequencias podem gerar confusão com códigos benignos. Nos experimentos, foram comparadas três estratégias de uso de HMM: uso de limite, duelo e o esquema proposto pelos autores. Na maioria dos casos, o modelo proposto é mais rápido que as outras duas estratégias, também apresentando acurácia superior (0.98).

Canfora et al. (2016) testaram a eficiência dos modelos ocultos de Markov (HMM) para detectar famílias de *malware*, utilizando como características de 3 a 5 estados HMM. A partir da extração de sequências de instruções que melhor representam uma aplicação, os *opcodes* são convertidos utilizando o código smali/baksmali. Então, as sequências de *opcodes* são concatenadas em observações únicas. O conjunto de amostras utilizadas para os experimentos contêm 5560 aplicações confiáveis e 6192 *malware* para Android. Como resultado, constatou-se que HMM com 5 estados ocultos foi a melhor opção para diferenciar aplicações confiáveis de *malware*, com acurácia de 0,96. Já para identificar a família do *malware*, a entropia estrutural apresentou o melhor resultado, com precisão de 0.98.

Pai et al. (2016) utilizaram técnicas de agrupamento para classificação de *malware*, a partir de modelos ocultos de Markov, algoritmos com *K-means* e o de Maximização de Expectativas (EM). Para os

experimentos foram analisados *malware* de 4 famílias diferentes, totalizando mais de 8 mil exemplares, e um conjunto de programas genuínos. Como resultados, os índices obtidos pelo algoritmo EM foram melhores que os do *K-means*, melhorando a cada vez que o número de agrupamentos e de dimensões aumentava.

Ling e Sani (2017) desenvolveram uma curta revisão do uso de Modelos Ocultos de Markov (HMM) para detecção de *malware*, com foco em códigos metamórficos. De acordo com esse estudo, a principal forma de representação (de arquivos) utilizada é a de sequência de *opcodes*. Ainda ressalta-se que a seleção das características extraídas pode afetar a velocidade da análise e taxa de detecção quando usados os HMM.

Pitolli et al. (2021) desenvolveram o MalFamAware, para classificação automática de *malware* Windows em famílias. Foi utilizado o algoritmo de agrupamento BIRCH online, comparando o método desenvolvido com algoritmos de supervisionados e não-supervisionados. Em algumas situações, os autores relataram que os algoritmos de agrupamento EM e DBSCAN obtiveram maiores valores de acurácia, com 86% e 89% respectivamente, em comparação à acurácia de 81% do BIRCH. Em relação à velocidade, o BIRCH foi mais veloz que a maioria dos algoritmos de agrupamento executados, chegando a ser 15x mais rápido que o K-means.

3.3 REPRESENTAÇÃO DE DADOS E CLASSIFICAÇÃO DE *MALWARE*

Na pesquisa de Reddy e Pujari (2006) foi abordado o uso de *n-grams* para detecção de *malware*. Mediante demonstrações empíricas, os autores comprovaram que *n-grams* carecem de consciência semântica. Eles descreveram uma nova medida de seleção de atributos baseada na frequência de documentos de classes. Para isso, foram combinados métodos como SVM, árvores de decisão e IBK por meio do WEKA, utilizando a Teoria de Evidência Dempster-Shafer.

Gurrutxaga et al. (2008) analisaram o comportamento de duas representações de *malware* executáveis, comparando um conjunto de doze medidas de similaridade e três variações de algoritmos de *clustering* hierárquico. Os autores concluíram que, para o conjunto de exemplares, a melhor forma de ligamento foi a Ligação Média (*Average Linkage*). Já em relação às distâncias aplicadas, os autores concluíram que a comparação entre elas é mais complexa, entretanto ainda é possível descartar medidas que tiveram piores desempenhos em relação às outras do conjunto.

Vinod et al. (2012) também desenvolveram pesquisas aplicando conceitos de bioinformática à detecção de *malware* metamórficos. Existe uma técnica para alinhamento de sequências chamada MSA (*Multiple Sequence Alignment*). Essa forma de comparação tem sido eficaz no uso com sequências de proteínas e DNA. Nesse trabalho, os autores utilizam sequências de *opcodes* que podem ser considerados pontos de mutação, quando ocorrem mudanças de uma geração de código para outra. Após a criação da matriz de alinhamento, constrói-se uma árvore filogenética utilizando o método de Junção de Vizinhos (*Neighbour Joining*). A melhor taxa de detecção foi de 91% com taxa de falsos positivos chegando a 52%, significando que diversas amostras de softwares benignos foram classificadas erroneamente como *malware*.

Faruki et al. (2012) utilizaram *call-grams*, sequências geradas a partir de grafos de chamadas de API. Inicialmente, esses grafos são gerados a partir da captura da sequência em que essas APIs são chamadas pelo binário. Essa técnica resultou na diferenciação de códigos maliciosos de benignos com precisão de 98,4%, utilizando o método da Floresta Randômica (*Random Forest*).

Oprisa et al. (2014) apresentaram um método de agrupamento baseado em árvores sufixadas. Como características, foram utilizadas *strings* de códigos de operações (*opcodes*) extraídos dos binários. Foram implementados diversos algoritmos baseados em agrupamentos clássicos, como o de ligação simples, e no algoritmo de árvore sufixada generalizada de Ukkonen. Além disso, foi proposta uma medida de distância entre *opcodes*, denominada distância Deq. A partir dos resultados dos experimentos com 100 mil exemplares, acredita-se que o consumo de memória seja um ponto fraco do algoritmo.

Awad e Sayre (2016) apresentaram o uso de fluxo de controle estruturado (SCFG) para agrupamento automático de *malware*. Os SCFG foram gerados pela ferramenta comercial de engenharia

reversa estática Hyperion. Para estimar a similaridade entre as amostras, foram utilizadas *strings* de distância Edit entre as representações dos SCFG gerados. Foi aplicado um limite de qualidade de agrupamento (*Quality Threshold Clustering* – QTC), o qual agrupa objetos similares baseado em um valor limite pré-definido, que controla a máxima distância permitida entre dois elementos dentro de um *cluster*. Os autores ressaltam que mesmo sendo caro computacionalmente, o uso de um QTC é benéfico porque não exige o cálculo antecipado de um número de agrupamentos. O número de amostras utilizadas nos três experimentos é pequeno: 303 *malware* Windows. Acurácia média dos experimentos com o sistema foi de 94%.

Singh et al. (2017) utilizaram múltiplas sequências de chamadas a APIs em extração de *n-grams* e para geração de grafos de controle de fluxo (CFG). Foram analisadas 15 mil amostras de *malware* e 4000 arquivos benignos. A partir dos arquivos coletados, foram construídos conjuntos de dados de *2-grams*, *3-grams* e *4-grams* das sequências de chamadas a APIs . Sob os conjuntos de *n-grams*, foram aplicados algoritmos de aprendizagem de máquina, como *Random Forest*, j48 e *Naive Bayes*, para distinção de arquivos benignos de maliciosos. A maior taxa de acurácia obtida foi de 95%, na segunda fase de experimentos, a partir de aprendizagem cumulativa.

Silva et al. (2018) aprimoraram algoritmos de detecção de *malware*, utilizando informações de ícones de arquivos PE. Diversos algoritmos de agrupamento foram aplicados, sendo que os melhores resultados foram com os métodos baseados em densidade, DBSCAN e HDBSCAN. Na extração de características, destaca-se o uso de histogramas de gradientes orientados (HOG), com total de 576 características obtidas por ícone analisado. Além dessas características, foram adicionadas outras geradas através de *autoenconder*, totalizando 1.114 características por arquivo. Como resultado obteve-se melhora em 10% na acurácia na predição de *malware*.

3.4 PESQUISA DE CLONAGEM DE CÓDIGO

Ji et al. (2008) utilizaram conceitos do estudo evolutivo de códigos fontes para detecção de plágio. Os autores criaram árvore evolutiva de mais de 100 códigos-fonte, podendo detectar trechos plagiados com maior acurácia e confiabilidade. Outra contribuição deste trabalho é a determinação de uma medida de distância evolutiva, para inferir o grau e a direção da evolução entre os códigos-fonte analisados.

Li et al. (2010) apresentaram resultados preliminares para uso de detectores de plágio e outros desafios envolvendo análise de agrupamento de programas maliciosos. Os autores testaram três técnicas: APISeq, SYS3Gram e API3Gram para agrupamento de *malware*. Os resultados foram comparados com o popular algoritmo de agrupamento *BCHKK-algo*. Os autores ressaltam que a ideia principal do trabalho foi a sugestão de novas possibilidades para agrupamento de *malware*. Devido às novas técnicas de evasão que estão surgindo dos códigos maliciosos mais sofisticados, as técnicas de agrupamento e classificação de *malware* se tornam ineficientes cada vez mais rapidamente.

3.5 RECONSTRUÇÃO DA LINHAGEM

Ma et al. (2006) apresentaram um método para inferir a árvore evolucionária de famílias de *exploits*. Foram extraídos os *shellcodes* de 5 famílias de *malware* para análise. Isso justifica-se por ser o primeiro lugar onde o autor geralmente executa um código novo no sistema. Para comparação de pares extraídos dos *exploits* foram utilizadas medidas relativas de "distâncias edit" ao longo dos bytes dos *shellcodes*, abreviadas como "Exedit". Outras técnicas como a medida de distância Edit e uma versão de Grafo de Controle de Fluxo (CFG) foram utilizadas, mas demonstraram diversas limitações e erros ao não distinguir dados de código, também diversas vezes agruparam famílias distintas no mesmo conjunto de *exploits*. Para provar a metodologia, os autores realizaram a análise filogenética manual das famílias, confirmando a robustez da técnica apresentada no trabalho.

Hashimoto e Mori (2008) desenvolveram a Diff/TS: uma ferramenta para análise de mudanças estruturais feitas entre revisões de programas. Foram utilizados algoritmos de comparação de árvores com

controle conduzido por heurística configurável para várias linguagens de programação. Como experimento da ferramenta, os autores realizaram testes com filogenia de *malware* (*Sdbot worm*) baseada em controle de fluxo (CFG) e distância Edit. A ferramenta foi capaz de demonstrar uma área marginal na identificação do *malware*, demonstrando a consistência dos resultados da análise com os *clusters* identificados por inspeção manual. Conforme os resultados apresentados pelos autores deste trabalho, verifica-se que do mesmo modo que acontecem mudanças estruturais entre versões em programas benignos, acontece o mesmo para programas maliciosos. Os criadores de *malware* realizam alterações, criando variantes de famílias conhecidas, com objetivo de escapar do detector de assinaturas conhecidas dos softwares antivírus.

Dumitras e Neamtiu (2011) observaram linhas de evolução de software de código aberto para criar modelos de linhagem e proveniência de *malware*. No estudo apresentado combinaram métodos experimentais que usam conjuntos representativos de *malware* e informação contextual, reunidas a partir de hosts finais e *honeypots*. Árvores filogenéticas, técnicas de análise de séries temporais e de mineração de dados foram empregadas na construção da linhagem. Os autores ressaltam que assim como em outras subáreas da Segurança da Informação, existem diversas questões relacionadas à validação dos modelos, quanto à construção, contexto, além de validação interna e externa.

Vlachos et al. (2011) estudaram o impacto de agrupamentos filogenéticos de *malware* "na natureza". De acordo com as pesquisas realizadas pelos autores, apenas *malware* bem escritos, que apresentam facilidades para alteração/evolução, são os que permanecem no cenário como ameaças recorrentes. Como trabalhos futuros, a pesquisa visa prever ameaças iminentes através de modelos econométricos, a partir de dados de similaridades entre códigos e periodicidade de ameaças de famílias conhecidas.

Singh et al. (2012) também estudaram populações de *malware* ao longo do tempo. Para isso, utilizaram o conceito de rastreamento . Foram acompanhadas mais de 3500 exemplares de três famílias, ao longo de cinco anos. Para extração de características dos binários, foram utilizadas sequências mnemônicas de de *2-grams*. Os autores apontam que uma limitação deste estudo é o uso de apenas características estáticas. Ressaltou-se a necessidade de estudos futuros da relação existente entre desempacotamento de binários, evolução e rastreamento de *malware*.

Walenstein e Lakhotia (2012b) comentaram sobre a imprecisão comum entre modelos de relacionamento de *malware* e suas variantes. O *framework* proposto neste trabalho trata, uniformemente, da produção e evolução de códigos maliciosos, a partir de composições de transformações de código, distinguindo evoluções disjuntas e intercaladas. Os cenários abordados abrangem linhagens múltiplas com descendentes separados, compartilhamento de código intercalado, propagação de código entre as variantes de um *malware*, mecanismos de geração de código e de mudanças após a compilação, compartilhamento de características geradas e de suas atualizações, além de evoluções independentes do conjunto de ferramentas utilizadas. Nas conclusões, os autores ressaltam que acreditam que os trabalhos existentes apenas "arranham a superfície" da área de recuperação de proveniência e filogenia de *malware*, devido à complexidade dos problemas envolvidos.

Pfeffer et al. (2012) estudaram o processo evolutivo de *malware*, extraindo informações "genéticas". Foram utilizadas diversas técnicas de engenharia reversa, a partir da extração de características como informações críticas do cabeçalho PE, traços evolucionários e as denominadas funções linguísticas de binários. As funções linguísticas, apontadas pelos autores , tratam-se de intenções ocultas das atividades de comunicação, podendo dar suporte nas buscas de significados de alguns comportamentos maliciosos dos códigos. Além de análises dinâmica e estática, foram realizadas análise de comportamento e de funções, possibilitando a extração de informações da linhagem da família, em algumas situações.

Jang et al. (2013) visaram inferir linhagem de binários, benignos ou malignos, automaticamente. Os algoritmos propostos são de dois tipos: linha reta e de gráfico acíclico dirigido (DAG). Foram extraídas características dinâmica e estaticamente. Os autores ressaltam a importância de métricas para mensurar a qualidade dos resultados da linhagem. Para isso, usam 5 métricas para medir a acurácia do uso de gráfico acíclico dirigido e duas métricas para a técnica de linha reta. As métricas mais vantajosas foram a métrica de incompatibilidade de ordem parcial e a distância Edit dos arcos dos grafos. A acurácia média do sistema proposto foi de 84% para códigos benignos e 72% para códigos maliciosos.

Darmetko et al. (2013) ressaltaram a importância da extração da linhagem de *malware* para a Forense Computacional. Uma das contribuições dos autores demonstram como extrair e representar evidências de mudanças estruturais entre variantes. Os experimentos com as amostras do Projeto Cyber Gnome tiveram melhores resultados quando comparados a com *malware* encontrados "na natureza". Para a determinação da relação ascendente/descendente foi utilizado o motor Jess e um sistema de manutenção da verdade (*Truth Maintenance System*) . Dificuldades foram encontradas na extração da linhagem de amostras que foram empacotadas com diferentes empacotadores ou com técnicas avançadas de ofuscação.

Jilcott (2015) apresentou um sistema para classificação de *malware* em larga escala, mais voltado para investigação forense. O sistema, intitulado DECODE, utiliza análise estática, extraindo informações dos *malware* a partir de CFG dos binários e utiliza árvores filogenéticas para construir a linhagem evolutiva. O autor não detalha o processo de construção da árvore de filogenia, apenas garante que a criação desta taxonomia não depende de assinaturas de ataques de *malware* conhecidos. Além disso, ressalta que o sistema DECODE pode construir tal taxonomia a partir de um lote de *malware* nunca visto.

Bernardi et al. (2016) publicaram uma proposta de modelagem para construção de filogenias de *malware* para sistemas operacionais móveis, utilizando traços de chamadas de funções. Para isso, os autores utilizaram uma linguagem de modelagem de processo declarativo. Foram utilizadas 700 aplicações infectadas para testes. Além disso, os experimentos confirmaram os resultados já conhecidos da linhagem de 4 variantes da família DroidKungFu. *malware* e de seus atributos utilizando informação genética, denominado MAAGI. A partir de um modelo de *bag-of-words*, três tipos de características dos *malware* são usados como palavras: duas de sequências de caracteres e um tipo de *N-gram* com grafo de procedimento de chamadas. As primeiras sequências são extraídas a partir de sequências encontradas nos binários e nas bibliotecas usadas por eles. Já na parte de agrupamento, o método de Louvain, baseado em agrupamento de grafos, foi utilizado juntamente com o *K-means*. Além disso, apresentam um método para extração da linhagem dos *malware* utilizando um modelo probabilístico. Desse modo, como resultado os autores acreditam que o sistema é o primeiro a integrar agrupamento, linhagem e predição de *malware* e famílias.

Cozzi et al. (2020) analisaram o relacionamento de *malware* que atacam dispositivos de Internet da Coisas (IOT). Foram investigadas 93 mil amostras, coletadas em um período de três anos e meio. Foram utilizados grafos para agrupamento baseado em similaridade e reuso de código. Os autores estudaram a linhagem de duas grandes famílias, mediante características estáticas e dinâmicas.

3.6 ANÁLISE DE COMPORTAMENTO

Willems et al. (2007) abordaram análise de comportamento de *malware*. Foram analisados 6.148 exemplares, coletados em uma janela de 5 meses. De acordo com os autores, mais de 95% dos *malware* adicionam chaves no registro do sistema para habilitar mecanismos de auto execução. Além disso, outro comportamento malicioso comum citado no trabalho é a cópia que os binários realizam de seus arquivos para o diretório do Windows. Esses padrões mencionados definem comportamento suspeito, podendo ser utilizado por ferramentas, como o CWSandbox, para classificar programas em benignos e malignos.

Bailey et al. (2007) apresentaram uma nova técnica de classificação automática de *malware* que os descreve a partir de mudanças do estado do sistema. Após demonstrarem que os antivírus comerciais classificam ou nomeiam incorretamente entre 20 e 49% dos *malware*, os autores utilizam o comportamento dos códigos maliciosos para agrupá-los em *clusters* similares. Foram analisados 3700 exemplares, agrupados pelo método de Ligação Simples, utilizando NCD como métrica de distância.

Wagener et al. (2008) publicaram um novo método de extração de comportamento de *malware*. As principais características da técnica residem no uso de alinhamento de sequências para calcular similaridades em conjunto com cálculos de distâncias Hellinger. Os autores capturaram dados de um *honeypot* e objetivaram classificá-los automaticamente, com base apenas em seu comportamento. Para isso, ressalta-se que não foram utilizados métodos de desmontagem (*disassembly*) ou depuração (*debugging*). A execução do *malware* foi feita em uma máquina virtualizada, controlada e parada por uma heurística de tempo (10 segundos). As mensagens de execução *Raw* foram recuperadas via SSH, visando encontrar as chamadas de sistema executadas pelo código do programa malicioso. O *layout* da memória foi reconstruído

para determinar quais pedaços estavam relacionados à máquina virtual e quais estavam relacionados à execução das chamadas de sistema executadas pelo *malware*. Os autores utilizam os valores das distâncias para criar as árvores de filogenia, foram descobertas automaticamente 3 famílias diferentes de *malware*. A taxa de erro apresentada pelo autor foi de 7%.

Jiang e Zhu (2009) construíram a plataforma vEye para detecção de assinaturas de *worms*. A ferramenta proposta tinha como objetivo capturar perfis e comportamentos dinâmicos de infecções de *worms*, como sondagem, exploração e propagação. Foram utilizadas técnicas de mineração de dados e algoritmos de alinhamento de sequência, como por exemplo, o alinhamento de Smith–Waterman. Criou-se árvore de Filogenia para categorizar as sequências de assinaturas e guiar o alinhamento de múltiplas sequências que foram utilizadas, com foco nas raízes das "pegadas" do *worm*. Os resultados demonstraram eficiência na detecção de *worms* no mundo real. Entretanto, diversos problemas foram encontrados na avaliação da ferramenta utilizando *malware* com comportamentos polimórficos e também na detecção de vermes com características de camuflagem.

Perdisci et al. (2010) também apresentaram um agrupamento baseado em comportamento. A similaridade do tráfego HTTP gerado pelos *malware* foi utilizada como base em comparação estrutural. Foi implementado também um sistema de prova de conceito e testado com experimentos com mais de 25 mil exemplares distintos de *malware*, confirmando a eficiência da ferramenta proposta. Entretanto, ressalta-se que uma limitação desse trabalho, mais voltado para o comportamento do *malware* que aparece no tráfego HTTP, deve-se disparar eventos específicos para chamar atenção e ter traços diferentes, assim como pode ter problemas com criptografia no caso de programas maliciosos escritos para usar o protocolo HTTPS.

Rieck et al. (2011) desenvolveram um *framework* para análise automática de comportamento de programas maliciosos, utilizando técnicas de aprendizagem de máquina. O objetivo dos autores era permitir tanto o agrupamento quanto a classificação de amostras desconhecidas. Os experimentos realizados, com mais de 33 mil amostras, demonstraram significativa melhora no tempo de computação quando comparadas análise e agrupamento simples e análise incremental de traços de comportamento.

Neugschwandtner et al. (2011) também estudaram análise baseada em comportamento e desenvolveram a ferramenta Forecast. Uma das inovações apresentadas neste trabalho foi predizer em qual classe de comportamento um *malware* se encaixaria, antes da execução do arquivo malicioso. Foram utilizadas funções de ranqueamento para análise dinâmica em larga escala. Para a extração de características foram utilizadas informações de seções, rótulos de antivírus, nível de empacotamento, peHash, DLLs importadas, etc. Depois das características extraídas estaticamente, é feito o agrupamento de previsão do *cluster*, depois passando para análise dinâmica e agrupamento baseado em comportamento. Foram investigadas 600 mil amostras, demonstrando que os resultados foram 134% mais eficientes se comparados a estratégias aleatórias de seleção de características.

Park et al. (2013) propuseram um método para construir grafos de comportamento comuns para representar famílias de softwares maliciosos. São utilizados traços de chamadas como características dos comportamentos. A técnica proposta tem duas partes: na primeira, é gerado um grafo de comportamento para cada amostra do conjunto. Na segunda parte, os grafos de *malware* da mesma família são agrupados em um único grafo, que representa o comportamento genérico de todos os membros da família. Os autores concluem que o sistema tem custo de execução relativamente baixo, não precisando de um grande conjunto de amostras para realizar a detecção de *malware*.

Perdisci et al. (2013) desenvolveram um sistema escalável para agrupamento, a partir de análise de comportamento de *malware* baseados em HTTP. O principal objetivo do trabalho era investigar como gerar automaticamente assinaturas de rede, a partir dos agrupamentos resultantes. Características estruturais e estatísticas foram extraídas e utilizadas em dois níveis de agrupamentos hierárquicos incrementais. Foram feitos experimentos com mais de 65 mil amostras. Os autores observaram melhora no desempenho computacional na análise de grandes bases de *malware*, quando compararam com o sistema desenvolvido em (Perdisci et al., 2010).

Shi et al. (2014) propuseram um método para detecção e classificação de *malware* utilizando GHSOM – Mapa hierarquizado de auto-organização crescente. Foram utilizadas técnicas de mineração de

dados a partir de arquivos DLLs usados pelos *malware*. Nesse tipo de classificação utilizando GHSOM, amostras similares são agrupadas em classes e os mais semelhantes entre si, em subclasses. Os autores também sugeriram uma nova forma de medir a similaridade entre classes de diferentes árvores ou em diferentes ramos de uma árvore. O método proposto foi comparado com o de alguns antivírus comerciais, demonstrando taxas de detecção semelhantes e até superiores. Como futura proposta de trabalho, os autores indicam análise de comportamento dos *malware* em conjunto com o uso de DLLs.

Kumar e Kaur (2016) desenvolveram um mecanismo avançado para agrupamento de *malware* metamórficos e polimórficos a partir de agrupamento híbrido. Neste trabalho foram aplicadas técnicas baseadas em permissão e assinatura, a partir de análise de comportamento dos códigos maliciosos. Com base nas bibliotecas e funções executadas, o algoritmo KNN e métodos de detecção de padrões foram empregados para classificação de *malware* polimórficos e metamórficos. De acordo com os autores, o método demonstrou ter acurácia de 97,83%. Entretanto, não foi mencionado o número total de amostras analisadas.

Ming et al. (2016) abordaram a vulnerabilidade de análise atual de comportamento de *malware*. As especificações de comportamento atuais podem ser atacadas por técnicas de ofuscação. A principal diferença do método proposto é a substituição de grafo de dependência das chamadas de sistema por variantes semânticas equivalentes. Na pesquisa foi utilizado o algoritmo LSH baseado em agrupamento hierárquico de ligação simples. Foram utilizados 5 conjuntos de dados diferentes, que continham códigos originais, com substituição de 10% a 30% das sequências, variantes artificialmente geradas, etc. Apesar de abaixar o desempenho, os resultados com a substituição proposta foi significativa contra os ataques estudados.

Sisaat et al. (2017) utilizaram agrupamento de *malware* com um propósito diferente: agrupar famílias a partir de comportamento de *download*, por hora e dia. Foram agrupados os top 10 *malware/bots* mais baixados, entre 2010 e 2011. Os dados foram adquiridos a partir de 90 *honeypots* espalhados no Japão. Como trabalho futuro, os autores pretendiam identificar *clusters* de *malware* por país, a partir dos endereços Ips. Os resultados foram apresentados com precisão de 75% e *recall* de 86,7%. O conjunto de amostras utilizadas foi de 32 milhões de registros de *download* de códigos maliciosos.

Abdullah e Chanderan (2017) apresentaram os resultados preliminares de um *framework* para análise automática de *malware*, utilizando técnicas de agrupamento baseado em HDBSCAN. O método provê descobertas de casos atípicos, como análise de comportamento anormal e discriminação de variantes desconhecidas. Apesar de não apresentarem dados de experimentos e taxas de acurácia e *recall* no momento, os autores esperam que o *framework* desenvolvido permita análises rapidamente.

3.7 ANÁLISE DE ATAQUES

Filiol (2006) objetivou criar um sistema robusto contra análise de caixa preta. São utilizadas assinaturas e sub-assinaturas para representação do *malware*, utilizando modelo matemático. O esquema de detecção dificulta ações de *copycats*.

Wehner (2007) publicou um estudo utilizando técnicas de compressão e Complexidade de Kolmogorov para análise de *worms*. A compressão dos binários foi feita usando UPX (*Ultimate Packer Executables*) (Wehner, 2007). Uma das diferenças do trabalho de Wehner é o fato de usar apenas as diferenças de compressão, não utilizando padrões pré-selecionados ou características específicas de *worms* para analisar tráfego de rede. A autora explica que diferenças nas taxas de compressão de sessões de rede podem ajudar a distinguir diferentes tipos de tráfego, tornando mais simples a detecção de anomalias e ataques. No experimento foram utilizados 719 diferentes tipos de *worms*, binários e não binários. Apenas 13 exemplares não foram classificados corretamente na geração de modelos filogenéticos. Como resultado do trabalho foram disponibilizados dois *plugins* para o software detector de intrusão Snort.

Há alguns anos, trabalhos na área de segurança introduziram uma nova técnica para detecção de possíveis ataques contra algoritmos de agrupamento e seus possíveis impactos. Biggio et al. (2014a) comprovou que o algoritmo de agrupamento por Ligação Simples (*Single Linkage*) pode ser gravemente afetado pela presença de ataques de envenenamento bem elaborados nos dados de entrada. Biggio et al.

(2014b) fez a mesma análise com o algoritmo de Ligação Completa. A vulnerabilidade também se mostrou significativa com esse método de agrupamento, reforçando que esse pode ser um dos pontos fracos do sistema de segurança utilizado.

Lee et al. (2015) utilizaram técnicas de agrupamento para detectar grupos de *botnets* infectados pelo mesmo arquivo malicioso. Foram utilizadas URLs contidas em e-mails e em arquivos anexados, sendo que mais de 57 mil e-mails foram coletados para investigação. O esquema detectou que 85,36% dos ataques registrados eram de três tipos específicos: técnica direta-para-MX, campo de recebido falso e a técnica de retransmissão aberta. Os autores pretendem aplicar o algoritmo em um portal de sites, porém desejam suplementar o sistema antes. É desejável tornar o sistema escalável, por causa dos novos tipos de ataques que surgem a cada ano.

Guralnik et al. (2017) relataram que antivírus que utilizam agrupamento hierárquico não supervisionado estão sujeitos a "ataques de envenenamento". Nesse tipo de ataque, um usuário mal-intencionado pode degradar o desempenho do software antivírus ao submeter para o banco de dados amostras projetadas especificamente para colapsar a hierarquia de classificação utilizada pelo antivírus. Apresentam-se resultados na aplicação de uma nova noção de entropia para dendrogramas combinatórios ao problema de controlar o influxo de amostras na base de dados e desviar ataques de envenenamento. É realizada demonstração formal dos teoremas.

Moubarak et al. (2017) destacaram os principais aspectos de filogenia encontrados em ciberataques. Os autores comentam sobre o uso de *malware* como armas pelos governos e seus exércitos. Citam o uso do Stuxnet, um dos *malware* por eles estudado, para destruir centrífugas de Urânio. Para estudo da evolução e dos relacionamentos de *malware*, foram utilizadas comparações de fluxos de controle, com base em informações compartilhadas entre as variantes. A árvore filogenética de 8 *malware* foi apresentada, demonstrando o panorama das principais ameaças no Oriente Médio.

3.8 PROPOSTAS DE TAXONOMIA DE *MALWARE*

Seideman et al. (2014) introduziram o conceito de gênero de *malware*: um conjunto de programas maliciosos que consiste em amostras estritamente relacionadas, determinado pelas relações entre as amostras dentro da população do *malware*. São utilizados critérios de tempo, nome e características, extraídas dinamicamente, das funções executadas pelos códigos. Os autores ressaltam que enquanto os gêneros atendem os critérios estabelecidos, é importante notar que haverá exemplos de amostras de *malware* que aparentemente deve ser incluído dentro de um gênero, mas não deve. Isso pode acontecer quando existem amostras de *malware* dentro de uma família maior agrupada naquela família devido a algum critério de avaliação, mas que não é tão próxima quanto o esperado. Enquanto os dois gêneros podem fazer parte da mesma família geral, o fato de estarem separados mostra que ocorreu algum tipo de divergência na filogenia - pode ser nesse ponto em que várias novas amostras de *malware* são geradas.

Liles et al. (2015) apresentou uma pesquisa teórica relacionada a usar o poder de arma que os *malware* podem ter na sua taxonomia. Os autores criticam o uso do comportamento dos *malware* como fator principal na classificação das famílias. Eles ressaltam que esse tipo de classificação pode ser confusa devido ao fato que diversos tipos de *malware* apresentam funções e comportamentos semelhantes. Os autores questionaram a importância das características evolutivas e da filogenia na classificação dos *malware*. Verifica-se que a proposta de taxonomia de Liles et al. (2015) é interessante. Entretanto, por ser apenas um trabalho teórico e inicial, ficam várias questões sobre a viabilidade técnica da taxonomia. Além disso, conforme já apresentado por outros autores, a importância da rápida detecção do comportamento, funcionalidades, códigos similares e pontos fracos semelhantes permite aos analistas de segurança maior rapidez na disponibilização de métodos para conter os danos causados pelos *malware* aos usuários e empresas prejudicadas.

Chakraborty et al. (2017a) apresentaram uma ideia para classificação automática de arquivos maliciosos, através de uma estrutura hierárquica (tipo do *malware*, plataforma, família e variante). Através do estudo de máquinas hierárquicas de vetores de suporte (SVM), definiu-se o uso do algoritmo de transferência ortogonal para classificação automática de arquivos maliciosos executáveis e portáteis. O

conjunto de *malware* utilizados para experimentos foi composto por 3.6 milhões, apresentando redução de 36% na taxa de erro de binários, em comparação a uso de estruturas não-hierárquicas de representação de arquivos.

3.9 TRABALHOS CORRELATOS

Hayes et al. (2008) propôs um *framework* para avaliação de modelos de filogenia de *malware*, em sua dissertação de mestrado. Para avaliação dos sistemas, foram criadas artificialmente variantes de *malware*, permitindo simulação de evolução e utilizando como métrica comparação de grafos filogenéticos. Os dois modelos propostos neste trabalho foram um modelo baseado em mutação e outro baseado em acréscimo de características .

Inteligência artificial tem sido utilizada há décadas para classificação e agrupamento de *malware*. Entretanto, Jordaney e Nouretdinov (2016) ressaltaram que, na maioria das vezes, os algoritmos são vistos como uma caixa preta, não sendo questionados caso apresentem grande acurácia. Mas, como mensurar a qualidade de uma amostra dentro do conjunto, por exemplo? Partindo disso, desenvolveram um *framework* para avaliação da qualidade dos resultados utilizando conceitos da estatística, como credibilidade e confiança. Dentro do sistema construído destacam-se duas técnicas de análise: uma para avaliar classificação baseada em similaridade e outra, para agrupamento. Na experimentação foram testados três algoritmos de detecção e classificação de *malware*. Os resultados do avaliador de conformidade desenvolvido demonstraram que bons resultados de aprendizagem de máquina, de acordo com as métricas tradicionais, podem resultar em desempenhos inconsistentes.

Zhou e Jiang (2012b) fizeram estudos de caracterização e evolução de mais de 1200 amostras, de 49 famílias diferentes de *malware* Android, coletadas entre 2010 e 2011. Os resultados mostraram que 86% das amostras coletadas trata-se aplicativos legítimos empacotados com *payloads* maliciosos.

A ferramenta DroidLegacy foi apresentada por Deshotels et al. (2014) para extração automática de assinaturas de famílias de *malware* Android. De acordo com os autores, para diferenciar códigos benignos de malignos, foram utilizadas chamadas API por estarem relacionadas ao comportamento principal dos *malware* para esse sistema operacional. Inicialmente, é realizada análise estática dos *malware* para encontrar as dependências e são gerados os grafos de dependências das classes. Após esse processo, são identificados os módulos, são extraídas suas características e seus conjuntos de chamadas API. A partir desse momento, as assinaturas de cada família já podem ser extraídas. Foram testados 1052 *malware* e 48 *apks* benignos. Como foi utilizada análise estática, as desvantagens das técnicas de ofuscação e transformações dos *malware* não são suportadas pelo DroidLegacy. Os autores ressaltam a importância da filogenia em seus trabalhos anteriores, mas não aplicaram/não construíram árvores evolutivas neste trabalho.

Suarez-Tangil et al. (2014a) são os criadores do DENDROID, uma abordagem para analisar e classificar *malware* Android utilizando técnicas de mineração de texto. O sistema apresentado no trabalho dos autores utiliza análise estática, extraindo estruturas de código (métodos) das classes dos aplicativos e "características estatísticas". Os autores utilizaram CFG e agrupamento hierárquico, assim como árvores evolutivas (dendrogramas) para classificação dos *malware* e das famílias analisadas. Os trabalhos futuros mencionados pelos autores sugerem o uso de semântica para investigação de *malware* com estratégias de ofuscação.

Eshghi et al. (2016) abordaram a diversidade existente na propagação de *malware* em redes móveis. Número de plataformas disponíveis, uso de lista de contatos, agrupamento na estrutura de rede, entre outros aspectos dificultam a contenção da disseminação dos programas maliciosos. Para tanto, os autores propuseram um *framework* formal para alavancar a heterogeneidade da rede, utilizando modelo de correção dinâmica, para obter o custo agregado mínimo devido à disseminação de *malware* e sobretaxa de correção (*patches*).

Asquith (2016) propôs um novo esquema para armazenamento eficiente e agrupamento de metadados de grandes conjuntos de *malware*. Foi utilizada uma técnica de redução de grafos, a qual apresentava vantagens na análise das redundâncias que eram criadas. A partir de um conjunto diverso de

características, foi possível simplificar a tarefa de agrupar amostras similares, mesmo em conjuntos de variantes polimórficas e/ou ofuscadas. O primeiro experimento envolveu 82 milhões de arquivos PE, sendo extraídas cerca de 193.6 características de cada arquivo. Foi possível reduzir as informações relacionais em 93%, a partir da redução de grafos. Já no segundo experimento, envolvendo mais de 2 milhões de arquivos *apks*, a taxa de redução de informação ficou em torno de 95%.

O trabalho de Canfora et al. (2016) também se relaciona ao estudo de filogenia de *malware* Android. A partir de uma análise estática, características foram extraídas de um *dataset* total com 4000 *malware*. Desse conjunto, 3000 foram utilizados para treinamento e 1000 para teste. Os métodos utilizados neste estudo foram: extração da distribuição relativa de frequência de *opcodes*, extração do CFG e análise de Isomorfismo utilizando *n-grams*. Os autores comentam que foi utilizada Filogenia, porém não apresentam detalhes da sua construção nem da árvore resultante. A taxa de eficácia do sistema proposto foi de 57%.

Hsiao et al. (2016) destacaram que são utilizadas diversas *threads* para executar tarefas intrinsecamente complexas em aplicativos Androids. Desse modo, torna-se difícil analisar aplicativos maliciosos sem conhecimento prévio de suas estruturas. A partir de análise de comportamento, os autores propuseram um esquema de investigação, o qual envolve árvores filogenéticas, UPGMA, componentes principais significativos e matriz de pontos. Para os experimentos foram estudadas mais de 980 amostras de diferentes famílias, sendo possível detectar qual código foi compartilhado entre aplicativos "irmãos" e qual foi herdado de ancestrais em comum.

Battista et al. (2016) apresentaram uma abordagem de detecção de famílias de *malware* Android, mediante análise de *bytecodes* Java, convertidos com uso do Cálculo do Sistema de Comunicação de Milner (CSS). Após a conversão JavaCode-CSS, os processos CSS são utilizados para estimar a lógica temporal de descoberta das famílias de *malware* para Android. A partir de um conjunto de 200 aplicações, 100 da família DroidKungFu e 100 amostras da família Opfake, foram geradas outras 200 aplicações modificadas, totalizando 400 amostras para experimentos. As taxas de acurácia dos experimentos variaram entre 83% e 94%.

Outro trabalho relacionado à análise de *malware* para Android foi apresentado por Cimitile et al. (2017). Diferentemente dos anteriores, CIMITILE et al analisaram os programas maliciosos dinamicamente a partir de chamadas de sistemas e também criaram suas árvores evolutivas, utilizando a ferramenta droidSapiens. Para os experimentos foram utilizados 858 exemplares agrupados em 5 famílias. Como trabalho futuro, os autores pretendem utilizar *k-bsimulation* para medir a similaridade entre as famílias de *malware* Android.

Raff e Nicholas (2017) desenvolveram uma nova forma de representação vetorial de características, aprimorando a distância LZJD - *Lempel-Ziv Jaccard Distance*. Os autores focaram em construir algoritmos eficientes tanto para treinamento (fraqueza do *n-grams*) quanto para inferência (fraqueza do LZJD). Um objetivo secundário era obter um classificador robusto para desequilíbrio de classes. O algoritmo proposto foi testado em conjunto de amostras de *malware* para Android e para Windows, sendo 2 milhões de dados de treinamento e 80 mil dados para testes. Os resultados, em comparação a outros algoritmos, como o SMOTE, apresentou acurácia significativamente superior enquanto reduzia a complexidade do algoritmo.

Martín et al. (2017) desenvolveram o MOCDroid, classificador evolucionário multiobjetivo para detecção de *malware* Android. Como características, foram utilizadas chamadas a um grupo específico de APIS - *"third party"*. Em conjunto a algoritmos genéticos, três algoritmos de agrupamentos foram utilizados: *K-means*, *PAM* e agrupamento baseado em modelo. Foram coletadas 9430 aplicações benignas e 9383 maliciosas para os testes. Os resultados, em comparação aos antivírus comerciais, foram 10% melhores, sendo que a acurácia foi próxima a 95%.

Li et al. (2017a) apresentaram uma técnica de agrupamento de *malware* Android através de mineração de *payloads* maliciosos. A solução proposta verifica se os aplicativos compartilham a mesma versão de *payload* malicioso. Os autores criaram um método para remover o código de bibliotecas legítimas de aplicativos Android, ainda mantendo os *payloads* maliciosos, mesmo que estejam injetados sob nome de bibliotecas populares. Em experimentos realizados, a taxa de acurácia foi de 0.90 e o *recall* foi de 0.75 no conjunto de *malware* Android Gnome.

Cheng et al. (2019) utilizaram grafos de chamadas a APIs para classificar variantes de famílias maliciosas conhecidas, com foco no sistema operacional Android. Para agrupar os grafos, os autores aplicaram os algoritmos Minimum Common Supergraph (MinSuper) e Maximum Common Subgraph (MaxSub) . Além disso, foi apresentado um protótipo, CuF, com detecção offline e treinamento online. O conjunto de amostras analisadas eram de seis famílias, totalizando 300 exemplares. Apesar da efetividade e da eficácia, os autores citam a desvantagem do *overhead* da geração dos grafos e da captura das chamadas a APIs dinamicamente. Em comparação ao trabalho de Kolbitsch et al. (2009), o *overhead* relativo foi 39,8% para o primeiro e 14,3% para o CuF .

Acharya et al. (2022b) aplicaram técnicas de transferência de aprendizagem, redes neurais convolucionais e algoritmos de agrupamento para detecção e classificação de *malware* Android. Nos testes realizados com os algoritmos de agrupamento ALD (Alocação Latente de Dirichlet) e agrupamentos hierárquicos, os últimos apresentaram melhores resultados na classificação em famílias. Porém, em relação aos valores de acurácia, o melhor resultado obtido do ALD foi de 98%, enquanto para o algoritmo de ligação média, a máxima foi de 79%.

3.10 *SURVEYS* ANALISADOS

Embora existam vários *surveys* sobre detecção de *malware*, essas revisões não são focadas em técnicas de agrupamento ou de filogenia. Na Tabela 1 são apresentadas pesquisas relacionadas à classificação de *malware* que foram estudadas nesta tese.

Tabela 3.2: *Surveys* relacionados à classificação de *malware*

Ref	**Escopo**	**S.O**	**Classificação**
Mohini et al. (2013)	Estudo sobre o modelo de segurança do Android, segurança no nível de aplicação e problemas de segurança da plataforma Android	Android	Binário
Gandotra et al. (2014)	Revisão das técnicas de análise, detecção e classificação de *malware*.	Windows	Binário
Feizollah et al. (2015)	Análise de seleção de características na detecção de código malicioso para sistemas operacionais móveis	Android	Binário
Arshad et al. (2016)	Revisão de diversas técnicas antimalware, seus benefícios e suas limitações.	Android	Binário
Riasat et al. (2016)	Um estudo sistemático baseado na detecção de *malware* Android de mercados de aplicações (*apps)* oficiais ou não.	Android	Binário
Baskaran e Ralescu (2016)	Uma investigação da evolução das técnicas de detecção de *malware* Android, focadas em algoritmos dde aprendizado de máquina.	Android	Binário
Yadav et al. (2016)	Esse estudo aborda o desenvolvimento das técnicas existentes de análise de *malware* Android.	Android	Binário
Malhotra e Bajaj (2016)	Uma revisão de técnicas de detecção de código malicioso focados em sistemas operacionais móveis.	Android, iOS e Symbian	Binário
Tam et al. (2017)	Uma revisão extensa sobre técnicas de análise e detecção de *malware* de diversos sistemas operacionais.	iOS, Windows OS, Palm, BlackBerry, Symbian e Android	Binário

Zachariah et al. (2017)	Esse *survey* é uma breve discussão sobre técnicas de detecção de *malware* Android.	Android	Binário
Yan e Yan (2018)	Os autores compararam métodos de detecção de *malware* para sistemas operacionais móveis, a partir de critérios de avaliação e métricas específicas.	Windows, iOS e Android	Multiclasse
Yu et al. (2018)	Um *survey* envolvendo descrições de comportamento de código malicioso, além de métodos de análise e de visualização.	Windows e Android	Binário
Odusami et al. (2018)	Os autores investigam técnicas de detecção de código malicioso para identificar lacunas na literatura.	Android	Binário
Hamed et al. (2019)	Essa revisão contém técnicas de detecção de código malicioso a partir de diferentes sistemas de detecção de intrusão baseados em nuvem.	iOS, Windows, Palm, BlackBerry,Symbian e Android	Binário
Alswaina e Elleithy (2020)	Os autores abordaram limitações encontradas na literatura, desafios e direções de pesquisas futuras no contexto de classificação de *malware* Android em famílias.	Android	Multiclasse
Naik e Dessai (2021)	Os autores analisaram pesquisas sobre detecção e classificação de *malware*, incluindo dados utilizados, extração de características e métodos de classificação. O foco principal do trabalho é o uso de aprendizado de máquina.	Windows e Android	Multiclasse
Dhalaria e Gandotra (2021)	Nessa revisão de literatura, são abordadas pesquisas de detecção de malware Android, principalmente as que utilizam aprendizado de máquina e deep learning. Um dos pontos ressaltados dos problemas encontrados relaciona-se a detecção de códigos maliciosos mais sofisticados.	Android	Binário
Gržinić e González (2022)	Além de métodos de classificação de *malware* Windows, os autores estudaram ferramentas de análise de código malicioso, com foco no desenvolvimento de métodos de análise automática.	Windows	Multiclasse

Outra pesquisa recente que aborda um problema crucial com os classificadores de malware do Android é a de Pendlebury et al. (2019). Os autores discutem como remover viés experimental na classificação de malware. Além disso, eles implementaram o TESSERACT, uma estrutura de avaliação de código aberto para comparar os classificadores de malware do Android em um ambiente real.

3.10.1 Filogenias baseadas em métodos de agrupamento

Tabela 3.3 apresenta informações dos artigos datados entre 2011 e 2015.

Tabela 3.3: Artigos analisados de 2011 a 2015 com informações sobre tipo de análise, características extraídas, uso de métodos supervisionados e não-supervisionados.

Ano		**2011**	**2012**	**2013**			**2014**					**2015**					
	Tipo/Referência	(Feizollah et al., 2018a)	(Wu et al., 2012)	(Aung e Zaw, 2013)	(Crussell et al., 2013)	(Samra et al., 2013)	(Shao et al., 2014)	(El Attar et al., 2014)	(Suarez-Tangil et al., 2014b)	(Feizollah et al., 2014)	(Deshotels et al., 2014)	(P.Faruki et al., 2015)	(Korczynski, 2015)	(Korczynski, 2015)	(Almin e Chatterjee, 2015)	(Chen et al., 2015)	Total
Análise	*Estática*		✓	✓	✓	✓	✓		✓		✓	✓		✓		✓	10
	Dinâmica	✓						✓		✓			✓		✓		5
	Híbrida																0
Características	*Permissões*			✓		✓								✓			3
	Chamadas a API										✓						1
	Opcodes																0
	Chamadas do sistema																0
	Dados de rede	✓								✓					✓		3
	Código				✓				✓			✓	✓			✓	5
	Outros		✓	✓			✓	✓									4
M.S	*K-means*			✓		✓		✓		✓				✓			5
	HAC						✓		✓				✓				3

	EM					✓		✓			✓						3
	Fuzzy	✓															1
	Birch														✓		1
	DBSCAN																0
	Modelo próprio				✓											✓	2
	Filogenia																0
	Outros						✓	✓		✓		✓					4
U.M	*Não menciona*	✓			✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	11
	SVM																0
	KNN		✓						✓								2
	Árvore de Decisão			✓													1
	RF			✓													1
	NB		✓											✓			2
	Logistica																0
	Neuro Fuzzy																0
	Outros																0

No período dos artigos da Tabela 3.3, houve aumento de 200% na quantidade de *malware* coletados entre 2012 e 2013, de acordo com relatório da Mcafee (Cruz et al., 2021). Com essa grande quantidade de volume de novas amostras para investigação, apesar das desvantagens da análise estática, observa-se que esse tipo de análise prevalece entre a mais aplicada. Analistas a preferem principalmente devido à limitação de tempo, restrições de poder computacional, número de ferramentas disponíveis para engenharia reversa e descompressão dos APKs.

Em relação aos recursos utilizados, destaca-se a análise de código, permissões e vinculação de dados de rede relacionados à análise dinâmica. Neste período, nos estudos analisados o uso de métodos supervisionados em conjunto com algoritmos de agrupamento ainda era sucinto.

A Tabela 3.4 apresenta informações sobre os artigos estudados de 2016.

Tabela 3.4: Estudos analisados de 2016 com informações sobre tipo de análise, características utilizadas, uso de métodos supervisionados e não-supervisionadoss

	Ano	**2016**								
	Tipo/Ref	(Dutta, 2016)	(Canfora et al., 2016)	(Hsiao et al., 2016) (Jang et al., 2016)	(Sartea et al., 2016)	(Verma e Muttoo, 2016)	(Bernardi et al., 2016)	(Asquith, 2016)	Total:	
Análise	*Estática*		✓				✓		✓	3
	Dinâmica	✓		✓	✓	✓		✓		5
	Híbrida									0
Características	*Permissões*						✓			1
	Chamadas a API			✓						1
	Opcodes		✓							1
	Chamadas do sistema					✓		✓		2
	Dados de rede				✓					1
	Code			✓						1
	Outros	✓			✓				✓	3
S.M	*K-means*						✓			1
	HAC			✓	✓	✓		✓	✓	5
	EM									0
	Fuzzy									0
	Birch	✓								1
	DBSCAN									0
	Modelo próprio									0
	Filogenia		✓							1
	Outros					✓				1
U.M	*Não menciona*	✓	✓	✓	✓			✓	✓	6
	SVM									0
	KNN									0
	Árvore de decisão						✓			1
	RF									0
	NB									0
	Logistica									0
	Neuro Fuzzy									0
	Outros					✓				1

Em 2016, notou-se um aumento na utilização do HAC, tanto com análise estática quanto dinâmica. Entre 2016 e 2017, houve um aumento na aplicação de técnicas de análise dinâmica, conforme pode-se visualizar na Tabela 3.4. Um fato que possivelmente contribuiu para esse aumento foi que algumas ferramentas popularmente utilizadas para análise dinâmica de *malware* Android foram publicadas nesse período, como o Droidmate (Jamrozik e Zeller, 2016) e o Droidbot (Li et al., 2017b). Outro fato importante para o período foi o lançamento da arquitetura multicamadas do Google, em uma revisão de segurança do Android em 2017. No entanto, como a maioria dos dispositivos ainda estava executando versões de sistema desatualizadas, as explorações das vulnerabilidades de aplicativos continuaram a se espalhar (Moses e Morris, 2021).

A Tabela 3.5 apresenta informações sobre os artigos estudados de 2017.

Tabela 3.5: Estudos analisados de 2017 com informações sobre tipo de análise, características utilizadas, uso de métodos supervisionados e não-supervisionados

	Ano	**2017**														
	Tipo/Ref	(Altaher, 2017)	(Canbek et al., 2017)	(Chakraborty et al., 2017b)	(Cimitile et al., 2017)	(Hamid et al., 2017)	(Hassen e Chan, 2017)	(Hurier et al., 2017)	(Kasiran et al., 2017)	(Li et al., 2017a)	(Meng, 2017)	(Milosevic et al., 2017)	(Pavlenko et al., 2017)	(Wang et al., 2017)	(Yang et al., 2017)	Total:
Análise	*Estática*	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓	12
	Dinâmica				✓									✓		2
	Híbrida															0
Características	*Permissões*	✓	✓	✓					✓			✓	✓		✓	7
	Chamadas a API										✓		✓		✓	3
	Opcodes															0
	Chamadas do sistema				✓		✓									2
	Dados de rede													✓		1
	Código										✓	✓				2
	Outros					✓		✓		✓			✓			4
S.M	*K-means*					✓			✓			✓	✓	✓		5
	HAC									✓					✓	2
	EM	✓										✓				2
	Fuzzy												✓			1
	Birch															0
	DBSCAN	✓									✓					2
	Modelo próprio		✓	✓			✓	✓								4
	Filogenia				✓											1
	Outros											✓				1
U.M	*Não menciona*		✓	✓	✓			✓		✓	✓		✓	✓		8
	SVM											✓			✓	2
	KNN														✓	1
	Árvore de decisão											✓			✓	2
	RF					✓						✓				2
	NB															0
	Logistica															0
	Neuro Fuzzy	✓														0
	Outros	✓					✓		✓			✓				4

Os dados dos artigos analisados de 2017 demonstram a tendência de análise de permissões de aplicativos fortemente ligada à análise estática. Ao visualizar a Tabela 3.5, a preferência por K-means permaneceu constante. Informações dos artigos de 2018 são apresentados na Tabela 3.6.

Tabela 3.6: Estudos analisados de 2018 com informações sobre tipo de análise, características utilizadas, uso de métodos supervisionados e não-supervisionados

	Ano	**2018**													
	Tipo/Ref	(Acampora et al., 2018)	(Atzeni et al., 2018)	(Chang et al., 2018)	(Fan et al., 2018a)	(Feizollah et al., 2018b)	(He et al., 2018)	(Ilham et al., 2018)	(Kim et al., 2018)	(Martinelli et al., 2018)	(Narayanan et al., 2018)	(Rathore et al., 2018)	(Shang et al., 2018)	(Xiong et al., 2018)	Total:
Análise	*Estática*				✓			✓			✓	✓	✓	✓	6
	Dinâmica	✓		✓		✓									3
	Híbrida		✓				✓		✓	✓					4
Características	*Permissões*							✓					✓		2
	Chamadas a API		✓	✓	✓				✓						4
	Opcodes	✓								✓		✓			3
	Chamadas do sistema										✓				1
	Dados de rede					✓	✓								2
	Código							✓							1
	Outros								✓	✓				✓	3
S.M	*K-means*					✓	✓	✓			✓	✓		✓	6
	HAC											✓			1
	EM														0
	Fuzzy	✓				✓									2
	Birch											✓			1
	DBSCAN		✓									✓			2
	Modelo próprio			✓					✓		✓	✓	✓		5
	Filogenia	✓			✓					✓					3
	Outros											✓			1
U.M	*Não menciona*	✓	✓	✓		✓	✓	✓	✓	✓					8
	SVM				✓							✓			2
	KNN				✓									✓	2
	Árvore de decisão				✓							✓			2
	RF				✓							✓			2
	NB											✓	✓		2
	Logistica											✓			1
	Neuro Fuzzy														0
	Outros										✓				1

Em 2018, o uso de análise estática e permissões como atributo continuaram predominantes. Nota-se na Tabela 3.5 o aumento no número de artigos combinando algoritmos supervisionados e não-supervisionados. A diversidade de algoritmos de agrupamento também aumentou, incluindo DBSCAN, Fuzzy e EM. Além disso, as chamadas de API foram utilizadas nos três tipos de análise de *malware*. A Tabela 3.7 apresenta os dados dos artigos datados de 2019.

Tabela 3.7: Estudos analisados de 2019 com informações sobre tipo de análise, características utilizadas, uso de métodos supervisionados e não-supervisionados

	Ano	**2019**													
	Tipo/Ref	(Fan et al., 2019a)	(Fan et al., 2019b)	(Ghosh e Mills, 2019)	(Lee et al., 2019)	(Khoda et al., 2019)	(Lou et al., 2019)	(Naeem et al., 2019)	(Saudi et al., 2019)	(Vega Vega et al., 2019)	(Yuan et al., 2019)	(Zhang et al., 2019b)	(Zhang et al., 2019a)	Total:	
Análise	*Estática*	✓	✓			✓			✓			✓	✓	✓	7
	Dinâmica			✓				✓			✓				3
	Híbrida				✓		✓			✓					3
Características	*Permissões*				✓		✓	✓				✓			4
	Chamadas a API		✓		✓		✓								3
	Opcodes				✓	✓								✓	3
	Chamadas do sistema			✓											1
	Dados de rede														0
	Código				✓								✓		2
	Outros	✓							✓	✓	✓				4
S.M	*K-means*	✓		✓	✓			✓	✓						5
	HAC											✓			1
	EM														0
	Fuzzy														0
	Birch													✓	1
	DBSCAN														0
	Modelo próprio		✓			✓	✓				✓		✓		5
	Filogenia									✓					1
	Outros	✓			✓		✓				✓	✓		✓	6
U.M	*Não menciona*		✓	✓		✓				✓		✓			5
	SVM				✓			✓	✓				✓	✓	5
	KNN	✓			✓				✓				✓		4
	Árvore de decisão	✓											✓		2
	RF	✓											✓		2
	NB				✓				✓				✓		3
	Logistica	✓													1
	Neuro Fuzzy														0
	Outros	✓			✓		✓				✓		✓		5

Observou-se que o desenvolvimento de novos *malware* Android diminuiu em 2018 (Institute, 2020). A partir do segundo semestre de 2019, o desenvolvimento de novas variantes aumentou de forma consistente. Nesse contexto, a análise dinâmica é interessante para estudar estruturas de códigos maliciosos e desenvolver vacinas. Pesquisas com análise dinâmica e híbrida aumentaram neste período (Tabela 3.7). Os artigos foram rotulados como Análise Híbrida ao usar técnicas estáticas e dinâmicas juntas.

A Tabela 3.8 apresenta dados dos artigos de 2020, enquanto a Tabela 3.9 apresenta dados dos estudos de 2021 e 2022.

Tabela 3.8: Estudos analisados de 2020 com informações sobre tipo de análise, características utilizadas, uso de métodos supervisionados e não-supervisionados

Ano		**2020**						
	Tipo/Ref	(Andresini et al., 2020)	(Appice et al., 2020)	(Ardimento et al., 2020)	(Cimino et al., 2020)	(Xu et al., 2020)	Total:	
Análise	Estática	✓	✓			✓	✓	4
	Dinâmica			✓	✓			2
	Híbrida							0
Características	*Permissões*	✓	✓					2
	Chamadas a API	✓	✓				✓	3
	Opcodes					✓		1
	Chamadas do sistema			✓	✓			2
	Dados de rede	✓	✓					2
	Código							0
	Outros	✓	✓					2
S.M	*K-means*	✓					✓	2
	HAC							0
	EM							0
	Fuzzy							0
	Birch							0
	DBSCAN							0
	Modelo próprio		✓			✓		2
	Filogenia			✓	✓			2
	Outros							0
U.M	*Não menciona*			✓	✓			2
	SVM						✓	1
	KNN							0
	Árvore de decisão		✓					1
	RF	✓						1
	NB							0
	Logistica							0
	Neuro Fuzzy							0
	Outros	✓				✓		2

Apenas visualizando os dados dos artigos analisados de 2020, pode ser complexo perceber uma tendência significativa. Mudanças constantes no cenário de *malware* impactam consequentemente essa área de pesquisa. Em 2018 e 2019, houve um surto de *ransomware* Android (Sharma et al., 2020). Apesar do número de ataques por esse tipo de *malware* ter diminuído em 2020, houve o início da pandemia do COVID-19. Com esse momento atípico, vários aplicativos maliciosos que usaram a doença como tema foram o foco dos cibercriminosos (Wang et al., 2021).

Tabela 3.9: Estudos analisados entre 2021 e 2022 com informações sobre tipo de análise, características utilizadas, uso de métodos supervisionados e não-supervisionados

	Ano	**2021**										**2022**		
	Tipo/Ref	(Katebi et al., 2021)	(Karbab e Debbabi, 2021b)	(Li et al., 2021)	(Liu et al., 2021)	(Taha e Malebary, 2021)	(Xiao et al., 2021)	(Zhao et al., 2021)	(Karbab e Debbabi, 2021a)	(Nomura et al., 2021)	(Rathore et al., 2021)	(Mosharrat et al., 2022)	(Kumar et al., 2022)	Total:
Análise	*Estática*	✓	✓	✓		✓	✓	✓	✓	✓	✓			9
	Dinâmica											✓	✓	2
	Híbrida				✓									1
Características	*Permissões*	✓			✓	✓		✓		✓				5
	Chamadas a API	✓		✓	✓			✓		✓				5
	Opcodes				✓		✓		✓		✓			4
	Chamadas do sistema													0
	Dados de rede												✓	1
	Código		✓	✓				✓						3
	Outros	✓			✓		✓	✓	✓			✓	✓	7
S.M	*K-means*	✓			✓						✓	✓	✓	5
	HAC				✓						✓			2
	EM													0
	Fuzzy				✓	✓								2
	Birch				✓						✓			2
	DBSCAN	✓	✓	✓		✓			✓		✓			6
	Modelo próprio	✓						✓						2
	Filogenia									✓				1
	Outros	✓	✓		✓		✓			✓	✓			5
U.M	*Não menciona*				✓									1
	SVM						✓	✓			✓		✓	4
	KNN							✓			✓		✓	3
	Árvore de decisão						✓				✓	✓		3
	RF	✓					✓	✓		✓	✓	✓	✓	7
	NB						✓	✓					✓	3
	Logistica						✓	✓						2
	Neuro Fuzzy													0
	Outros	✓	✓	✓		✓	✓	✓	✓		✓		✓	9

A partir de 2018, notou-se um aumento consistente no uso de chamadas de API para classificar *malware* Android. Estudos indicam que os *trojans* têm conjuntos maiores de chamadas de API em comparação com outras categorias de *malware* (Mirzaei et al., 2019). Além disso, de acordo com relatórios de empresas de soluções anti *malware*, os cavalos de Tróia continuam sendo as maiores ameaças aos consumidores em 2021, representando 90% de todas as ameaças relacionados à pandemia (Castillo et al., 2014).

3.10.2 Discussão

Assim como a quantidade de novas variantes e famílias de *malware* para Android aumentaram nos últimos anos, pesquisas sobre agrupamento e filogenia estão presentes em periódicos e empresas de antivírus. Os resultados da análise dos estudos estão resumidos a seguir.

***Datasets* e ferramentas**: Sobre datasets, existem várias fontes disponíveis gratuitamente para pesquisa, conforme disponível na Tabela 2.2. No entanto, embora disponibilizem vetores de características prontos para análise, raramente apresentam informações detalhadas sobre o conjunto de exemplares, amostras categorizadas por tipo de *malware*. Além disso, o número de amostras disponíveis ainda não é representativo e não é atualizado continuamente (Tam et al., 2017).

Número de amostras: Em relação ao número de exemplares de *malware* usados em experimentos, 15 dos estudos analisados utilizaram entre 100 e 1000 amostras, 11 estudos entre 1000 e 2000, dez estudos entre 5.000 e 10.000 e dez estudos entre 10.000 e 20.000 exemplares. Apenas três estudos analisaram mais de 1 milhão de espécimes. Mais informações sobre a distribuição do número de amostras entre os artigos são apresentadas na Figura 3.2.

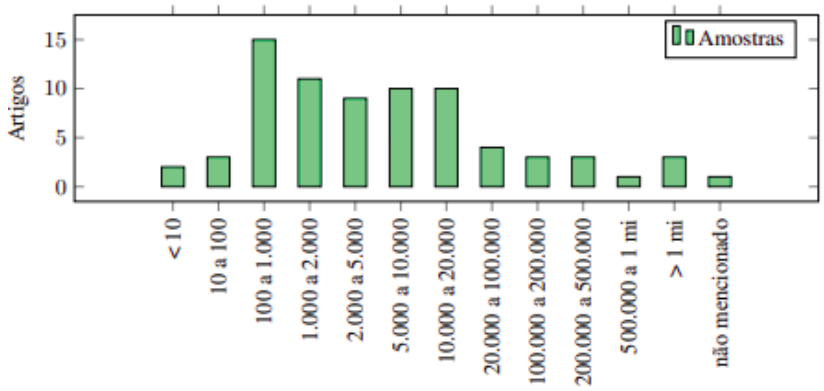


Figura 3.2: Número de amostras analisadas

Tipo de análise e uso de ferramentas: Em relação à análise de *malware*, 47 estudos aplicaram métodos estáticos, 20 utilizaram análise dinâmica e 8 aplicaram análise híbrida. No caso de estudos de filogenia apenas 60 % utilizaram análise dinâmica. Menos de 50 % dos estudos de filogenia de *malware* focados na plataforma Android analisados apresentaram informações ou discussões sobre as relações entre as famílias analisadas. As ferramentas mais citadas pelos autores estão na Figura 3.3, com destaque para Apktool e Weka.

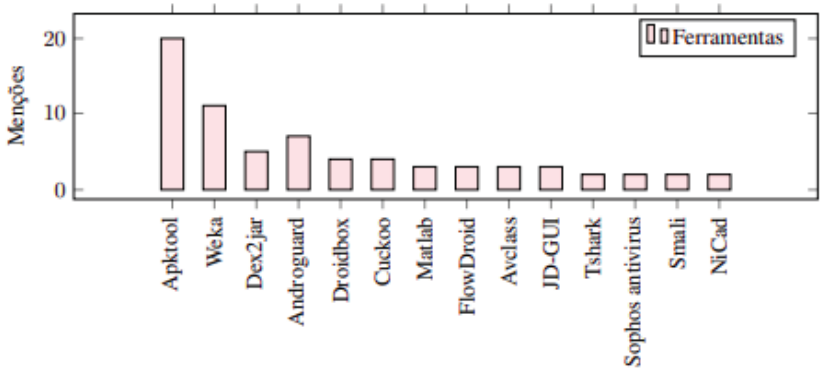


Figura 3.3: Ferramentas mais populares nos artigos analisados.

Características e representação de dados: Como pode ser visto na Figura 3.4, as características mais aplicadas foram análise de permissões e de chamadas a APIs. Em ordem decrescente de uso, os métodos incluem vetores binários, arquivos XML, grafos, *strings*, n-grams, arquivos ARR, cadeias de Markov, imagens em escala de cinza e arquivos no formato XES.

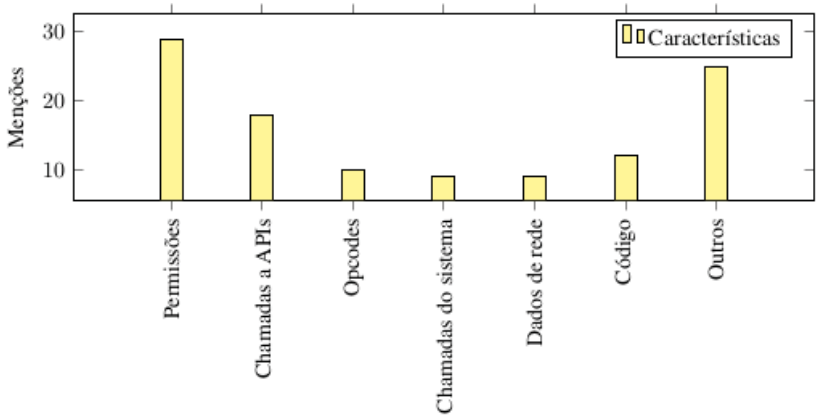


Figura 3.4: Características mais populares nos artigos analisados.

Medidas de distância e de similaridade: Na literatura de agrupamento de códigos maliciosos foi encontrada grande variedade de medidas de distância e de similaridade. Entretanto, as medidas mais utilizadas foram a distância euclidiana e o Índice de Jaccard. Além disso, o número de estudos que não mencionaram a medida utilizada também foi significativo (15 estudos em 82).

Métodos não-supervisionados mais frequentes: como mencionado anteriormente, os métodos de agrupamento foram aprimorados nos últimos anos. No entanto, mais de dez estudos analisados empregaram o algoritmo K-means como método primário de agrupamento (Figura 3.6). Além disso, alguns trabalhos executaram agrupamento aliado a outras técnicas de Inteligência Artificial, como pode ser visto na Figura 3.5. Dentre essas técnicas, destaca-se o uso de Random Forest e SVM.

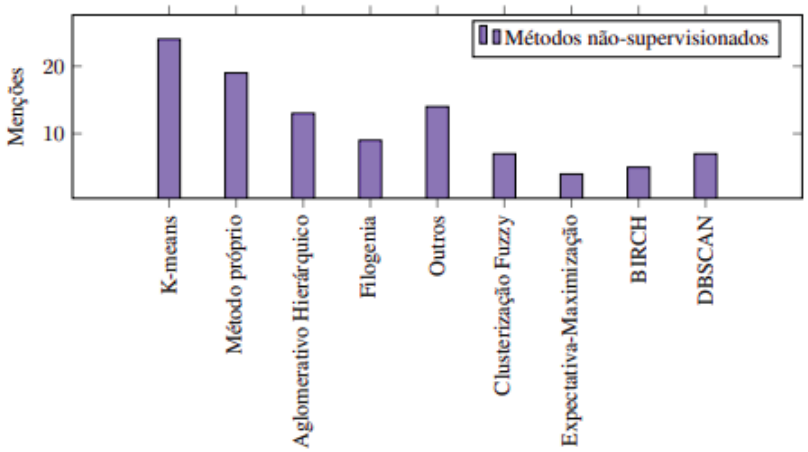


Figura 3.5: Métodos não-supervisionados mais frequentes nos artigos estudados.

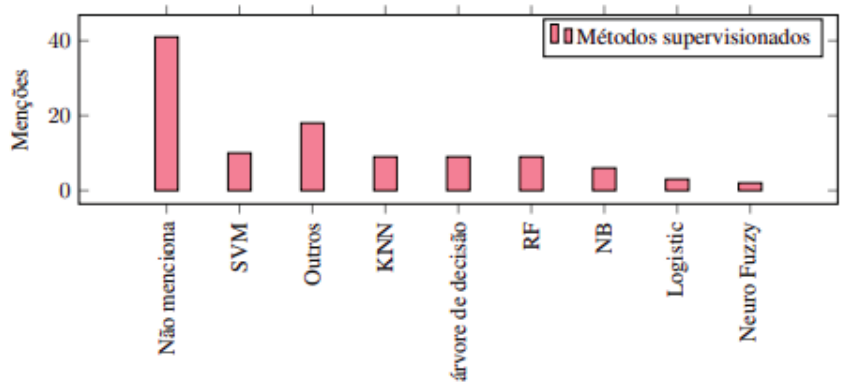


Figura 3.6: Uso de métodos supervisionados em conjunto com agrupamento.

Validação de Resultados de agrupamento: Sobre os estudos analisados que citaram índices de validação de resultados, 33% citaram Acurácia, 17% citaram F-Measure e 16% citaram índices de Verdadeiro Positivo e Falso Positivo. Pureza, pontuação de Calinski Harabaz, curva ROC e validação de outros índices também foram mencionados.

Notou-se falta de normalização e de padrões na apresentação e validação dos resultados, principalmente em estudos de filogenia que não utilizaram métodos populares de agrupamento.

3.11 CONSIDERAÇÕES SOBRE A REVISÃO

O presente capítulo apresentou informações sobre os principais trabalhos desenvolvidos nos últimos anos, sobre filogenia e classificação de *malware* Android em famílias.

Sobre os trabalhos estudados destaca-se:

- Diversos trabalhos aliaram o estudo de linhagem de código, versões de software e plágio com evolução e linhagem de códigos maliciosos (Ma et al. (2006), Hashimoto e Mori (2008), Dumitras e Neamtiu (2011), Pfeffer et al. (2012), Jang et al. (2013), Darmetko et al. (2013), Jilcott (2015), Bernardi et al. (2016)).

- Diversos trabalhos utilizaram técnicas de Inteligência Artificial, como mineração de dados, em conjunto com métodos não-supervisionados. (Almin e Chatterjee (2015),

Altaher (2017), Andresini et al. (2020), Appice et al. (2020), Aung e Zaw (2013),Fan et al. (2018b), Fan et al. (2019a),Hamid et al. (2017), Hassen e Chan (2017).

- Entre os artigos que citaram o uso de métodos hierárquicos de agrupamento, o Método *Average Linkage* (Ligação Média) foi o método que demonstrou os melhores resultados (Gurrutxaga et al., 2008).

Algumas lacunas percebidas no estado da arte incluem:

- Na literatura encontram-se estudos comparando eficiência de algoritmos de agrupamento (Yim e Ramdeen (2015), Suyal et al. (2014), Pai (2015)) e de classificação supervisionadas (Manzano et al. (2022), Shalaginov et al. (2018)), medidas de similaridade (Zhu et al. (2020), Pandit et al. (2011)) e formas de análise de binários (Kang et al. (2012)). Todavia, faltam análises do impacto de formas de representação de dados na classificação de exemplares.
- Ausência da padronização de passos relevantes na classificação não-supervisionada de *malware* em comparação com a ampla literatura na classificação supervisionada.
- Falta de padronização na avaliação da qualidade dos agrupamentos de família de *malware*: por exemplo, quais índices e métricas são indicados de acordo com estilos de agrupamento utilizados?

4 EXPERIMENTOS COM MEDIDAS DE SIMILARIDADE, ALGORITMOS DE AGRUPAMENTO E FLUXO DE ATIVIDADES

Após a revisão da literatura, foram realizados quatro experimentos iniciais para comparar o desempenho de medidas de similaridade/dissimilaridade e de algoritmos de agrupamento. Além disso, foi desenvolvido um fluxo de atividades para auxiliar em tarefas e decisões relacionadas a agrupamento de *malware*, podendo ser aplicados para a plataforma Android e para outros tipos de sistema operacional. O fluxo de atividades também é apresentado no presente capítulo.

4.1 EXPERIMENTO 1

Esse experimento foi realizado, inicialmente, para verificar a eficácia do conjunto de opções selecionadas: medidas de similaridade/dissimilaridade + características + tipo de ligação do algoritmo de classificação hierárquica. Além disso, teve como objetivo examinar a viabilidade da análise visual de famílias de *malware* através de dendrogramas.

Em relação ao conjunto de amostras, foram analisadas 100 aplicações maliciosas de sistema operacional Android. Os arquivos foram escolhidos aleatoriamente do conjunto OmniDroid (Martín et al., 2019a). Esse *dataset* disponibiliza arquivos (.csv), de características extraídas dinâmica e estaticamente, de centenas de aplicações maliciosas e não-maliciosas. O histograma dos 100 arquivos é apresentado na Figura 4.1, com os nomes e seus respectivos tamanhos em *kilobytes*.

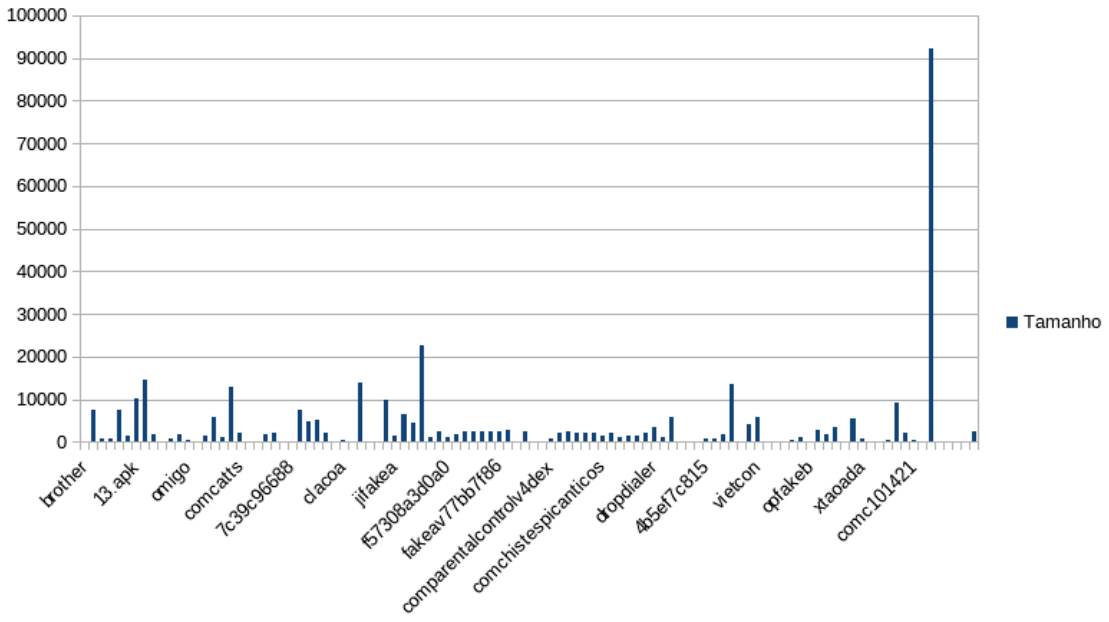


Figura 4.1: Histograma do conjunto com 100 amostras

Informações sobre as amostras maliciosas analisadas nesse experimento são apresentadas na Tabela 4.1.

Continua na próxima página

Nº	Nome do *Apk*	Tamanho	Descrição/Família
1	Vietcon	5,6 MB	TrojanDownloader/Artemis!
2	SamSapoA	473,9 KB	TrojanSMS/SamSapo
3	XtaoAdA	746,2 KB	TrojanDownloader, Artemis! ,Backdoor
4	SmsWorker	1,9 MB	Trojan.AndroidOS.Moavt.C!c
5	towelroot	101,5 KB	Towel-A, Exploit
6	SmsThief	175,8 KB	Trojan, A Variant Of Android/SMForw.RQ
7	OpfakeB	53,9KB	Android:OpFake-HK, Trojan
8	orgbenews	1,2 MB	Trojan,AndroidOS.Hoaber
9	OpfakeA	48 KB	Trojan-SMS.AndroidOS.Opfake.a
10	skype	5,8 MB	Trojan.AndroidOS.Rootnik
11	Dropdialer	1,8 MB	Trojan.AndroidOS.Fsm
12	comthinkking	2 MB	Android-Trojan/JsSMS.
13	OldbootA	53,7 KB	Trojan.AndroidOS.Oldboot.C
14	fakeAV9345...LabelReader	2,7 MB	Trojan.AndroidOS.Andef.C
15	f57308a3d0a09d0d..c1ad5f03	1,2 MB	Trojan.AndroidOS.Anubis.C!c
16	comrecipesmart	2 MB	Android-Trojan/JsSMS.3ce26
17	krepitmtdywtjexf-1	824,8 KB	Trojan.AndroidOS.Sacto.C!c
18			
19	ObadA-	84,3 KB	Android.Trojan.Obad.A
20	Omigo	597,9 KB	Android:Agent-CWQ
21	NotCompatibleA	22,8 KB	Trojan.AndroidOS.Nisev.C!c
22	fakeAV77BB...FC1LabelReader	2,5 MB	Trojan.AndroidOS.Andef.C!c
23	comprasesamor	1,4 MB	Android-Trojan/JsSMS.3ce26
24	fakeAV75B8...CFLabelReader	2,5 MB	Trojan.AndroidOS.Andef.C!c
25	fakeAV148B7...LabelReader	22 MB	Trojan[FakeAV]/Android.Andef
26	comstatetss	2 MB	Android-Trojan/JsSMS.3ce26
27	JiFakeA	1,3 MB	Trojan.AndroidOS.FakeInst.C!c
28	FakeValidation	12,6 MB	Trojan.AndroidOS.Opfake.C!cx
29	comkitchenn	2 MB	Android-Trojan/JsSMS.3ce26
30	f847b80ab00228af2e07e99709421d5...1d5	1,3 MB	Trojan.AndroidOS.Anubis.C!c
31	fakeAV1CA..1D2LabelReader	2,5 MB	Trojan.AndroidOS.Andef.C!c
32	Google-play	198,8 KB	Trojan.AndroidOS.Marcher.C!c
33	FakeBankB	229,8 KB	Trojan.AndroidOS.Gepew.C!c
34	mcpef	56,4 KB	Android-Trojan/Dowbro.fd19
35	comlaughtter	2 MB	Android-Trojan/JsSMS.3ce26
36	eba335956afad3b50a93...3a5720	2,8 MB	Trojan.AndroidOS.Anubis.C!c
37	Opfake	51,8 KB	Android-Trojan/FakeInst.88649
38	Fobus	123,5 KB	Android-Trojan/Fobus.8345
39	FakeCMCCA	217,6 KB	Android-Trojan/FakeInst.6c49
40	comparentalcontrolv4-dexguarded	833,6 KB	Trojan.AndroidOS.Generic.F!c9
41	GinMasterZAdvancedObfuscation	13,6MB	Android-Trojan/GinMaster.beb3
42	fakeAV36B17...7LabelReader	2,5 MB	Trojan.AndroidOS.Andef.C!c
43	Masnu	1,1 MB	Trojan.AndroidOS.Agent.C!c
44	fakeAV1E178E501B4...LabelReader	2,5 MB	Trojan.AndroidOS.Andef.C!c

45	e089ade5ea97a595....e431b91e4	954,3 KB	Trojan.AndroidOS.Anubis.C!c
46	comprasesfee	1,9 MB	Android-Trojan-JsSMS.3ce26
47	comromaticpost	1,4 MB	Android-Trojan-JsSMS.3ce26
48	fakeAV6F2..9CED92LabelReader	2,5 MB	Trojan.AndroidOS.Andef.C
49	comparentalcontrolv4-dexguarded	833,6 KB	Trojan.AndroidOS.Generic.F!c
50	FakeDoc	741,8 KB	Trojan.AndroidOS.FakeDoc.C!c
51	4ab8f26e...34	129,4 KB	Trojan.AndroidOS.Asacub.C!c
52	642da73...64e723	55,7 KB	Trojan.AndroidOS.Asacub.C!c
53	03122ade...c13777	5,2 MB	Android-PUP/Adload.ccd83
54	comimagepets	2 MB	Trojan.Android-PUP/Adload.ccd83
55	5622aac15c...864d2Ackposts	64,2 KB	Android/Hiddad.WB
56	2D66D7942148...21C89	39,2 KB	Android-Trojan/Pincer.5e37b
57	bed3e665d2...40ceb	7,2 MB	Android.Pincer.A
58	comcattss	2 MB	Android-Trojan/JsSMS.3ce26
59	comc101421042723	289,1 KB	Android-Trojan/JsSMS.3ce26
60	029758783d2...dcdc9b4	39,5 KB	Android-Trojan/JsSMS.3ce26
61	4f6146956b5....c2766898	1,9 MB	Trojan.AndroidOS.Generic.C!c
62	OldbootA	53,7 KB	Trojan.AndroidOS.Oldboot.C!c
63	BadNews-A	135,9 KB	Trojan.Android-Trojan/FakeInst.c587
64	fakeAV36B17791..7LabelReader	2,5 MB	Trojan.AndroidOS.Andef.C!c
65	Clash-of-Clans	90,1 MB	Trojan.AndroidOS.FakeDoc.C!c
66	0a8298d77996ec1...50e7808f82d60-	8,9 MB	Trojan.AndroidOS.Hiddad.C!c
67	4f6146956b50...df5804ac2766898	1,9 MB	Trojan.Exod.Android.7
68	0a8298d77996...b50e7808f82d60	8,9 MB	Trojan.AndroidOS.Hiddad.C!c
69	64ebe9b975....274e3eefd3ca7f24e350-	1,3 MB	Trojan.AndroidOS.Anubis.C!c
70	GinMasterZAdvancedObfuscation	13,6 MB	Android-Trojan/GinMaster.beb3
71	3e60b0f5...8edf4a5a3ebabd	13,1 MB	Android-PUP/Malct.d60bb
72	00d430877eed0...9f82f282af	224,6 KB	Trojan.AndroidOS.Mekir.C!c
73	BiosNativeMaliciousCode-	6,5 MB	Trojan.AndroidOS.Sadpor.C!c
74	ClacoA	310,3 KB	Trojan.AndroidOS.Ssucl.C!c
75	comfdhgkjhrtjkjbxmodel	2,5 MB	Android-PUP/Secad.8ed48
76	comfunnyys	2, MB	Android-Trojan/JsSMS.3ce26
77	comparentalcontrolv4	942,8 KB	Trojan.AndroidOS.Generic.F!c
78	e089ade5ea97a5...79d8f519cfe431b91e4	954,3 KB	Trojan.AndroidOS.Anubis.C!c
79	4593635ba7...157b5c4b1222e79909e838	7,2 MB	Trojan.AndroidOS.Agent.C!c
80	19162b06350310...67fad5f563c0270	7,2 MB	Trojan.AndroidOS.Piom.C!c
82	0806166....0436GlamorousSmoke	1 MB	Android-PUP/Airpush.80e39
83	7c39c9668...dabb42ee	741,8 KB	Android-PUP/Airpush.80e39
84	5781eb58f1e628930e3...1bb7e685c9c9	129,4 KB	Trojan.AndroidOS.Asacub.C!c
85	b5f9a2b92bccb21...8e5a79ed54025d	141,8 KB	Android:Banker-ADI [Trj]
86	f9a2b92b...f1a8e5a79ed54025d	141,8 KB	Trojan.AndroidOS.FakeDoc.C!c
87	574e59a...000e0	741,8 KB	Trojan.AndroidOS.FakeDoc.C!c
88	14d9f1a92dd984d6040cc41ed06e273e	166,9 KB	Trojan.AndroidOS.FakeDoc.C!c
89	aba17776b98b8660b...add6	9,6 MB	Trojan.AndroidOS.FakeDoc.C!c
90	96f95e432...51eac2	4,4 MB	Trojan.AndroidOS.FakeDoc.C!c
91	comchistescortos	1,8 MB	Trojan.AndroidOS.FakeDoc.C!c

92	3d3db692be45e...6b5c77e4	3,9 MB	Trojan.AndroidOS.FakeDoc.C!c
93	1264C25D67D....4A90F61F23	937,6 KB	Trojan.AndroidOS.FakeDoc.C!c
94	Blatantsms	4,8 MB	Trojan.AndroidOS.FakeDoc.C!c
95	6ba1de1c1b1294edf8...c976ca9ebc26b	4,9 MB	Trojan.AndroidOS.FakeDoc.C!c
96	598df...6bbd9	224,6 KB	Trojan.AndroidOS.FakeDoc.C!c
97	26fef238028...c9ea5f55	1,9 MB	Trojan.AndroidOS.FakeDoc.C!c
98	Agent	18,1 MB	Trojan.AndroidOS.FakeDoc.C!c
99	00e74c118fa3902e5c85fd8e37f3d084	18,7 KB	Trojan.AndroidOS.FakeDoc.C!c
100	0f5f1409b1ebbe....4e5	2 MB	Trojan.AndroidOS.FakeDoc.C!c

Os passos realizados no presente experimento podem ser visualizados na Figura 4.2.

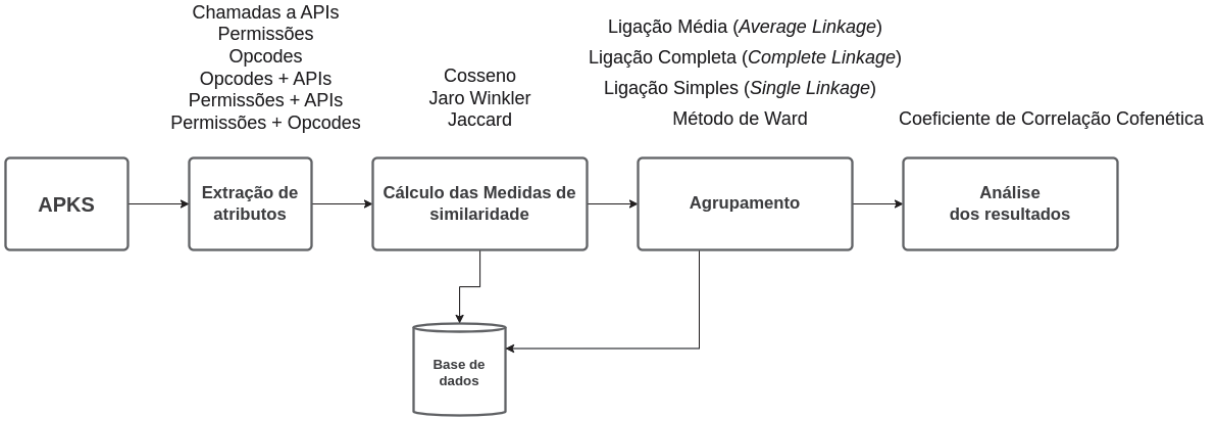


Figura 4.2: Método aplicado no Experimento 1

De acordo com a literatura estudada anteriormente, decidiu-se utilizar as seguintes características para classificação das famílias: chamadas a APIs, permissões utilizadas pelo aplicativo, *opcodes* e a concatenação das três par a par (apis-*opcodes*, apis-permissões, permissões-*opcodes*). Foram utilizados *4-grams* das sequências concatenadas. As medidas de similaridade aplicadas sob os *4-grams* foram : *Cosine*, Jaccard e Jaro Winkler. Como algoritmos de agrupamento foram empregados os seguintes: Método de Ward, Single, *Complete* e *Average linkage*, ou seja, Ligação Simples, Completa e Ligação Média.

A Tabela 4.2 apresenta os coeficientes de correlação cofenética calculados para os dendrogramas utilizando as sequências de APIs extraídas dos 100 *apks*.

Tabela 4.2: Coeficientes de correlação cofenética dos agrupamentos dos 100 *apks* - APIs

Método	*Cosine*	Jaccard	Jaro Winkler
complete	0,97464	0,92233	0,90503
Single	0,93457	0,47319	0,0,88559
Ward	0,96722	0,89700	0,90513
average	0,98354	0,88429	0,91829

Em relação à extração das chamadas a APIs, verifica-se que o maior coeficiente contido na Tabela 4.2 é o 0,97464. Da mesma forma que aconteceu em casos anteriores, esse coeficiente

é referente ao dendrograma utilizando a medida *Cosine* com o método de agrupamento *average linkage*.

A Figura 4.3 apresenta o dendrograma referente ao maior coeficiente cofenético (0,98354) da Tabela 4.2.

O dendrograma com as 100 amostras de *malware* já torna complexa a análise visual dos pares e de suas similaridades. No dendrograma apresentado na Figura 4.3, caso seja feita uma poda próxima à semelhança de 0.95, fica evidente a divisão em três grandes grupos de *malware*. O terceiro e último grupo, mais ao sul, possui os grupos de amostras com maior similaridade, sendo que o *cluster* verde possui mais similaridade que o *cluster* vermelho. Ainda pode-se verificar que a amostra 642da7. . . .723, próxima ao eixo x, apresenta grande dissimilaridade em relação aos outros 99 elementos da base estudada.

Considerando as famílias atribuídas aos *malware* (Tabela 4.1), percebe-se que os *malware* com índices 76,16,12,29,47 são da mesma família, Android-Trojan/JsSMS.3ce26, sendo classificados no mesmo *cluster* no dendrograma, o qual está destacado com a cor vermelha ao sul da imagem.

A Tabela 4.3 apresenta os coeficientes de correlação cofenética calculados para os dendrogramas utilizando as sequências de permissões extraídas dos *apks*.

Tabela 4.3: Coeficientes de correlação cofenética dos agrupamentos dos 100 *apks* - Permissões

Método	*Cosine*	Jaccard	Jaro Winkler
complete	0,66028	0,82287	0,86799
Single	0,69370	0,82084	0,59156
Ward	0,67052	0,85807	0,80003
average	0,67052	0,88752	0,87999

Em relação à extração das permissões, verifica-se que o maior coeficiente contido na Tabela 4.3 é o 0,88752. Da mesma forma que aconteceu em diversos casos anteriores, esse coeficiente é referente ao dendrograma utilizando a medida Jaccard com o método de agrupamento *average linkage*. O mesmo adequado foi o agrupamento Single *linkage* com a medida Jaro Winkler, com coeficiente de 0,59156.

A Figura 4.4 apresenta o dendrograma referente ao maior coeficiente cofenético (0,88752) da Tabela 4.3.

Considerando as famílias atribuídas aos *malware* (Tabela 4.1), percebe-se que os *malware* com índices 64,14,24 são da mesma família, Trojan.AndroidOS.Andef.C!, sendo classificados no mesmo *cluster* no dendrograma, o qual está destacado com a cor roxa, próximo ao centro da imagem. Além disso, os *malware* com índices 16, 25, 26, 29, da família Android-Trojan/JsSMS.3ce26, também aparecem no mesmo *cluster* (azul, no começo da imagem).

A Tabela 4.4 apresenta os coeficientes de correlação cofenética calculados para os dendrogramas utilizando as sequências de *opcodes* extraídos dos *apks*.

Tabela 4.4: Coeficientes de correlação cofenética dos agrupamentos dos 100 *apks* - *opcodes*

Método	*Cosine*	Jaccard	Jaro Winkler
complete	0,91184	0,82975	0,98071
Single	0,90020	0,73450	0,96840
Ward	0,92758	0,81811	0,97407
average	0,93812	0,85720	0,98237

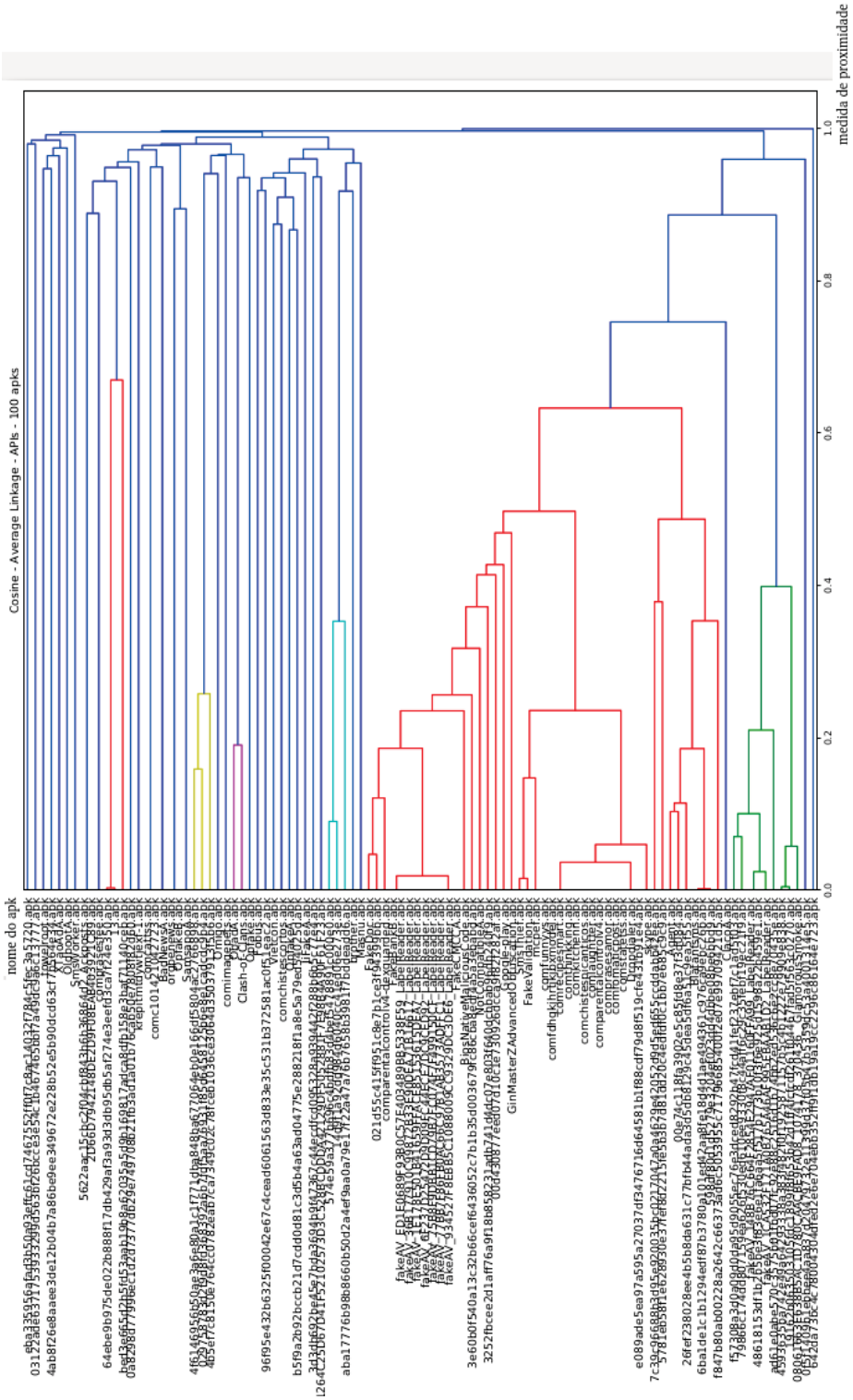


Figura 4.3: *Average linkage* – *Cosine* – 100 exemplares – APIs

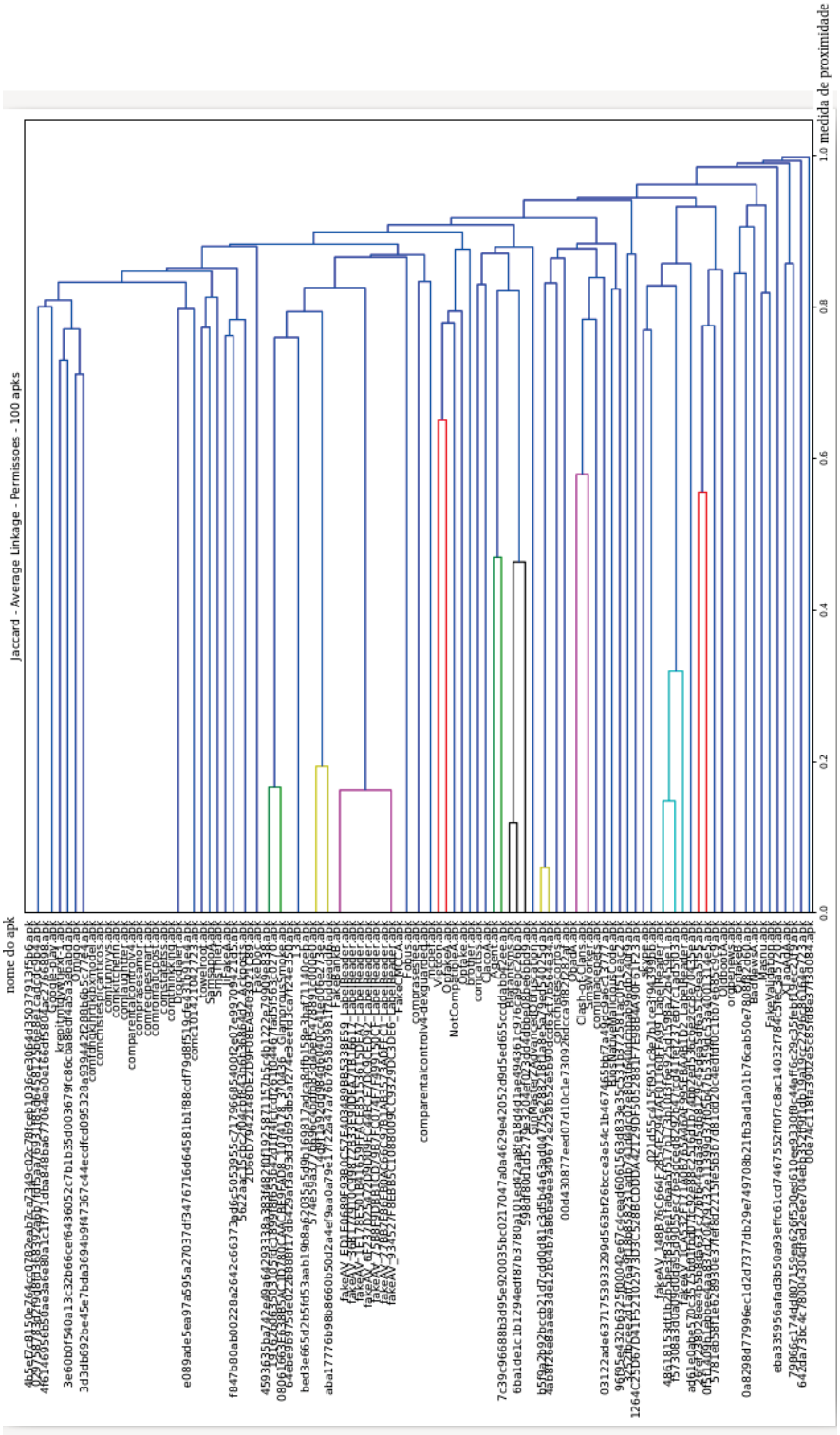


Figura 4.4: *Average linkage* – Jaccard – 100 exemplares – Permissões

A Figura 4.5 apresenta o dendrograma referente ao maior coeficiente cofenético (0,98237) da Tabela 4.4.

O agrupamento das variantes das famílias Android-Trojan/JsSMS.3ce26 e Trojan.AndroidOS.Andef.C! persistiu no dendrograma da Figura 4.5, assim como nos dois anteriores. Destaca-se ainda que os *malware* de índices 60 e 71 (Tabela 4.4), que são da família Android-PUP/Malct,também foram agrupados no mesmo *cluster*.

A Tabela 4.5 apresenta os coeficientes de correlação cofenética calculados para os dendrogramas utilizando as sequências de *opcodes* + permissões extraídas dos *apks*.

Tabela 4.5: Coeficientes de correlação cofenética dos agrupamentos dos 100 *apks* - *opcodes* + Permissões

Método	*Cosine*	Jaccard	Jaro Winkler
complete	0,90024	0,85288	0,94517
Single	0,84001	0,70040	0,93902
Ward	0,89057	0,81529	0,93969
average	0,91035	0,85067	0,94637

A Figura 4.6 apresenta o dendrograma referente ao maior coeficiente cofenético (0,94637) da Tabela 4.5.

Alguns agrupamentos das variantes das famílias Android-Trojan/JsSMS.3ce26 e Trojan.AndroidOS.Andef.C! persistiram no dendrograma 4.6. Entretanto, verifica-se que no caso de famílias como Trojan.AndroidOS.Agent.C!c e Trojan.AndroidOS.Generic.C!c, diversas variantes foram classificadas em *clusters* distantes.

A Tabela 4.6 apresenta os coeficientes de correlação cofenética calculados para os dendrogramas utilizando as sequências de *opcodes* + APIs extraídas dos *apks*.

Tabela 4.6: Coeficientes de correlação cofenética dos agrupamentos dos 100 *apks* - *opcodes* + APIs

Método	*Cosine*	Jaccard	Jaro Winkler
complete	0,97480	0,91239	0,90024
Single	0,93481	0,50046	0,89719
Ward	0,96766	0,89347	0,89217
average	0,98381	0,87843	0,92604

A Figura 4.7 apresenta o dendrograma referente ao maior coeficiente cofenético (0,98381) da Tabela 4.6.

Ao analisar-se o dendrograma da Figura 4.7, verifica-se que algumas variantes da família Android-Trojan/JsSMS.3ce26 foram classificadas em grupos distantes. Os *malware* de índice (4.1) 43 e 1, ambos da família Trojan.AndroidOS.Agent.C!c foram agrupados nos *clusters* ao norte do dendrograma, entretanto, não tão próximos quanto a variantes de outras famílias já mencionadas anteriormente.

A Tabela 4.7 apresenta os coeficientes de correlação cofenética calculados para os dendrogramas utilizando as sequências de permissões + APIs extraídas dos *apks*.

A Figura 4.8 apresenta o dendrograma referente ao maior coeficiente cofenético (0,97975) da Tabela 4.7. Nessa árvore verifica-se que diversas variantes, novamente, foram agrupadas juntos a outras da mesma família. Porém, diferentemente de agrupamentos anteriores, os *malware* de índices 3 e 43 foram agrupados no mesmo *cluster*. As variantes pertencem à família Trojan.AndroidOS.Agent.C!c.

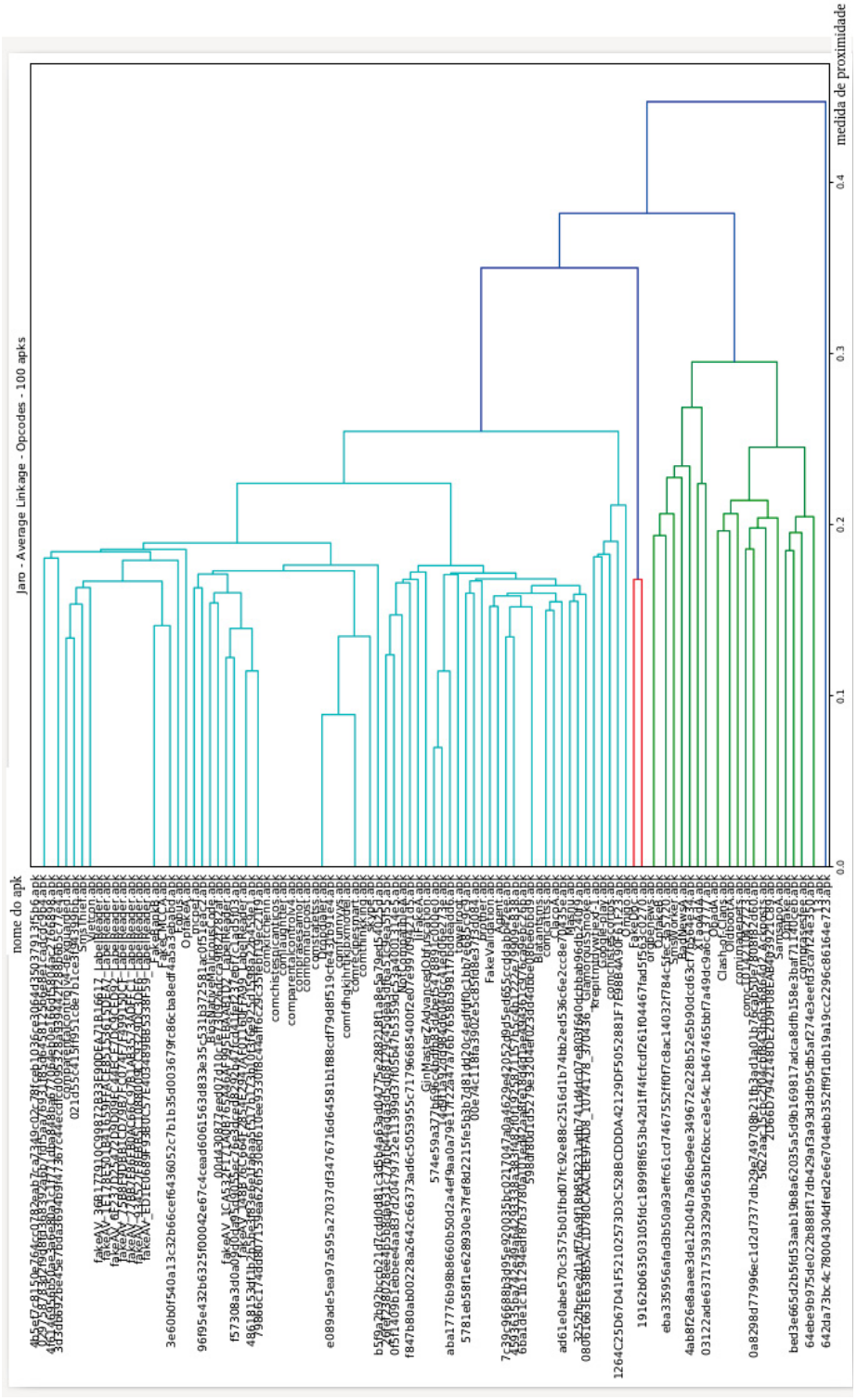


Figura 4.5: *Average linkage* – Jaro Winkler – 100 exemplares – *opcodes*

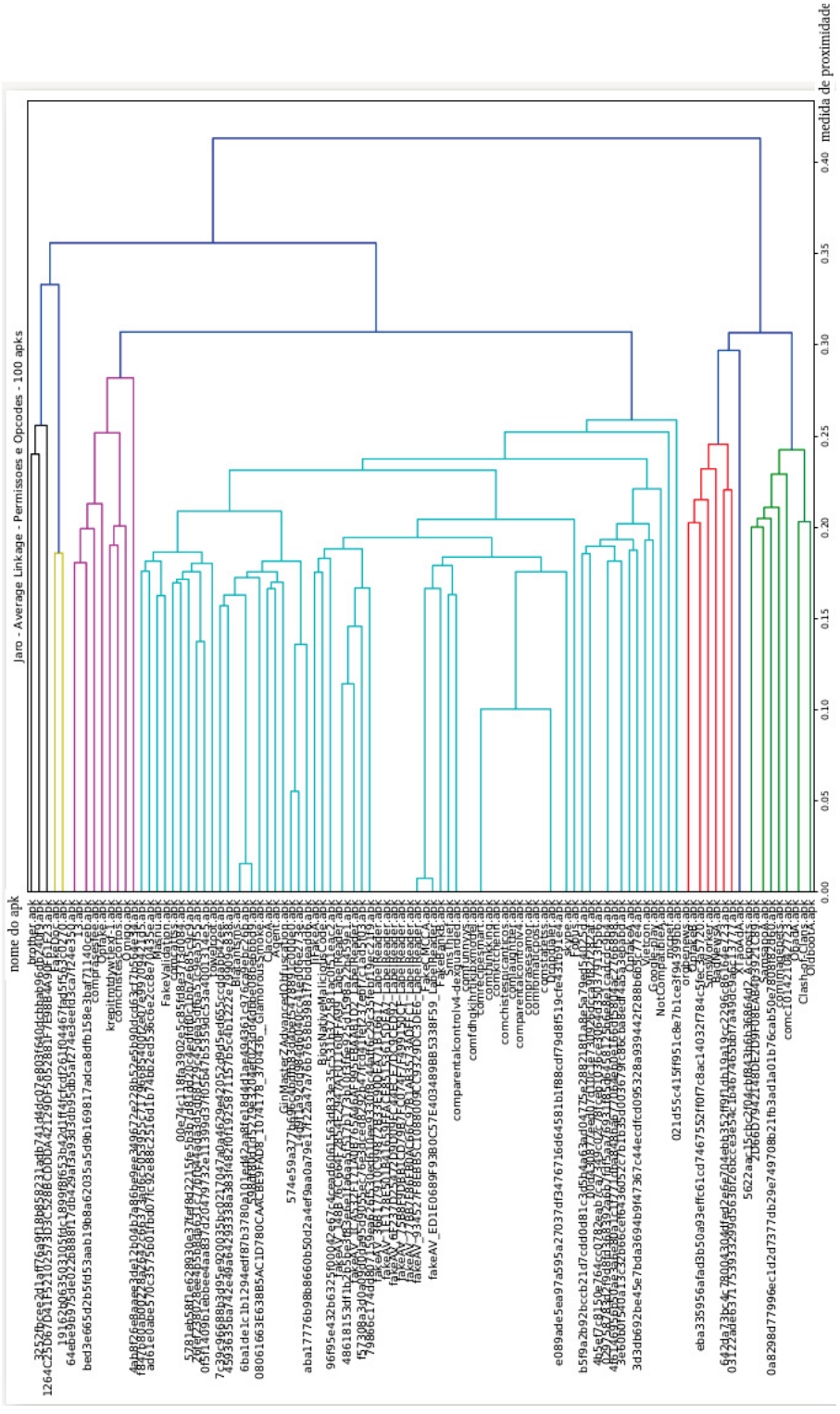


Figura 4.6: *Average linkage* – Jaro Winkler – 100 exemplares – *opcodes* + Permissões

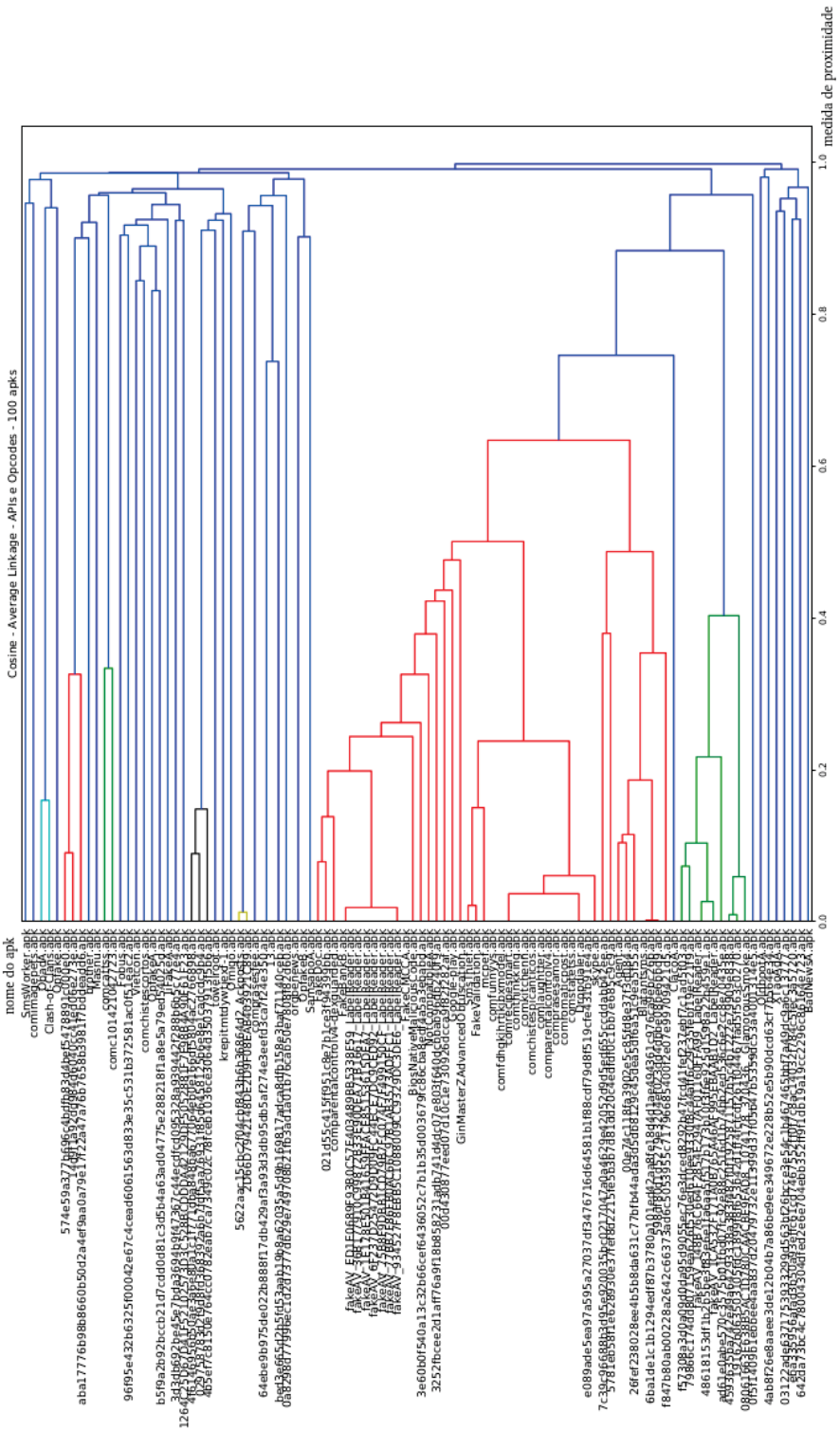


Figura 4.7: *average linkage* – *Cosine* – 100 exemplares – *opcodes* + APIs

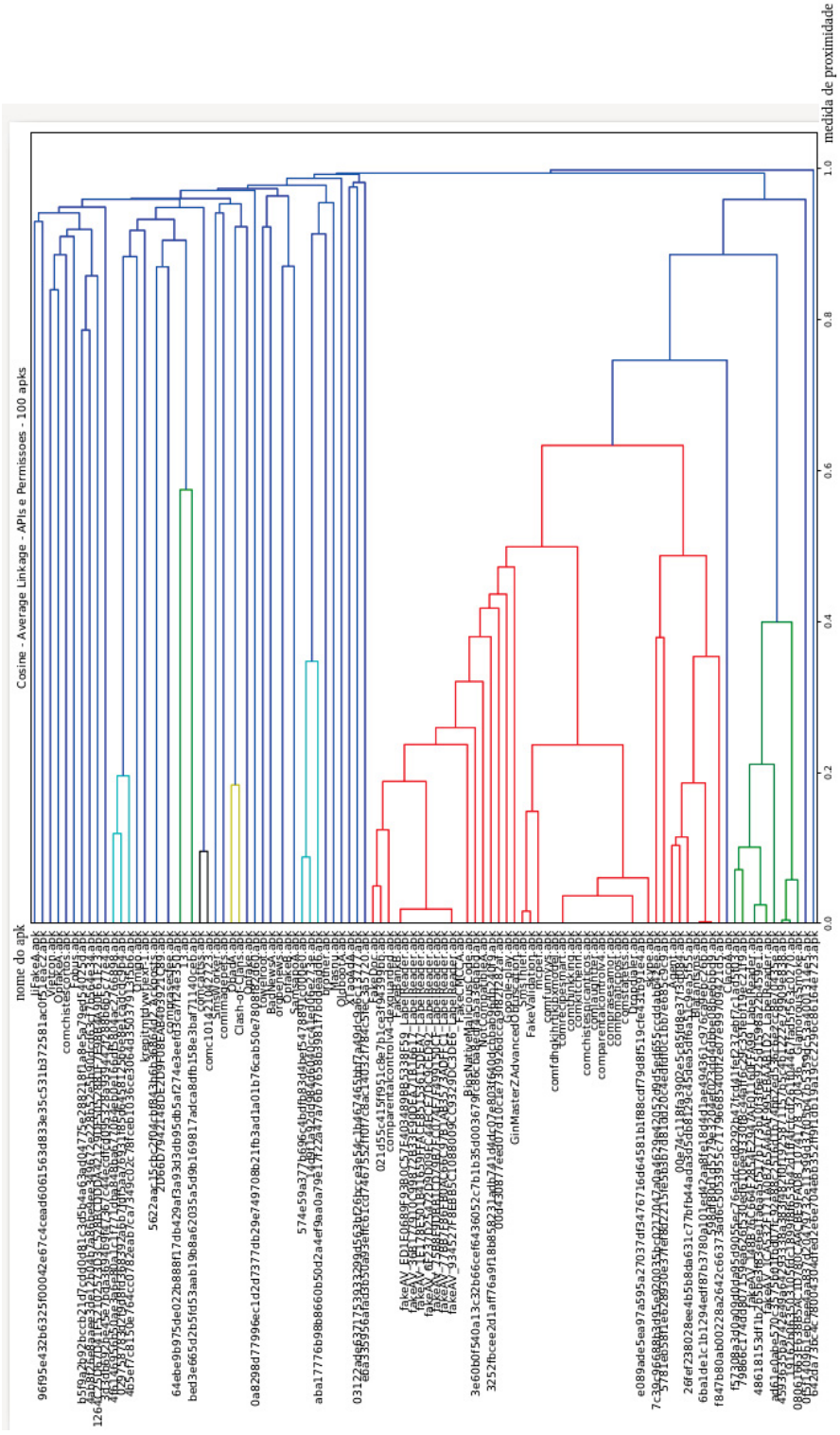


Figura 4.8: *Average linkage* – *Cosine* – 100 exemplares – Permissões + APIs

Tabela 4.7: Coeficientes de correlação cofenética dos agrupamentos dos 100 *apks* - Permissões + APIs

Método	*Cosine*	Jaccard	Jaro Winkler
complete	0,97457	0,92253	0,90501
Single	0,93791	0,47191	0,87728
Ward	0,96317	0,89632	0,91021
average	0,97975	0,88435	0,92264

Tabela 4.8: Coeficientes de correlação cofenética dos agrupamentos dos 100 *apks*. A segunda e a quarta coluna correspondem a combinação do tipo de ligação (*linkage*) e da medida de similaridade. A sigla C.C significa Coeficiente Cofenético

Características	Melhor conjunto	C.C	Pior conjunto	C.C
API	*Cosine* e Complete	0,98	Jaccard e *Single*	0,47
Permissões	Jaccard e *Average*	0,88	Jaro Winkler e *Single*	0,59
Opcodes	Jaro Winkler e *Average*	0,98	Jaccard e *Single*	0,73
Opcodes + Permissões	Jaro Winkler e*Average*	0,94	Jaccard e *Single*	0,70
API + Permissões	*Cosine* e*Average*	0,97	Jaccard e *Single*	0,47
Opcodes + API	*Cosine* e *Average*	0,98	Jaccard e *Single*	0,50

A Tabela 4.8 apresenta os melhores e piores resultados obtidos dos conjuntos de características, tipos de ligações e medidas de similaridade.

Ao analisar os coeficientes de correlação cofenética, verifica-se que a ligação média (*Average Linkage*) foi a mais apropriada em cinco das seis situações. Em relação aos desempenhos inferiores, a ligação simples (*Single Linkage*) predominou em todas as combinações. As medidas de similaridade obtiveram resultados semelhantes, todavia a medida Cosseno (*Cosine*) foi a melhor em metade dos cenários. Em relação às características utilizadas, o menor coeficiente foi apenas utilizando permissões, e os maiores relacionados a chamadas a APIs, opcodes e a combinação dessas duas características.

Em relação à análise visual dos dendrogramas gerados, destacam-se as seguintes ligações:

- As amostras SmsThief (índice 6) e Towelroot (índice 5) são as mais similares entre si. Analisando as chamadas a APIs, SmsThief possui 49, Towelroot, 8, de modo que 4 chamadas são iguais nas duas famílias.

- Ao se analisar as permissões extraídas dos apks, verifica-se que a variante OpfakeA possui registro de apenas 4 permissões, enquanto da variante OpfakeB foram extraídas 10 permissões existentes. Em vários dendrogramas, no mesmo *cluster* da OpfakeA encontrava-se a amostra Orgbenews, a qual possui 3 registros de permissões, as quais também pertencem ao conjunto do *malware* OpfakeA .Portanto, os algoritmos consideraram o *malware* OpfakeA com maior similaridade ao Orgbenews em comparação à OpfakeB.

- Outro par considerado com maior similaridade são os códigos XtaoAdA e SmsThief.O primeiro exemplar possui um conjunto de 21 permissões extraídas e o código SmsThief possui 8 permissões, das quais 4 permissões são comuns ao conjunto do XtaoAdA.

- Os dendrogramas resultantes do método de ligação média (*Average Linkage* com o índice de Jaccard para as características das chamadas a APIs e da concatenação de

APIs com permissões são semelhantes. Em ambos os dendrogramas, os elementos Towelroot e SmsThief , Orgbenews e OpfakeA foram agrupados nos mesmos *clusters*.

- Ao analisar-se o dendrograma do conjunto Cosseno + opcodes + chamadas a APIs com ligação média 4.7, verifica-se que algumas variantes da família Android-Trojan/JsSMS.3ce26 foram classificadas em grupos distantes. Desse modo, verifica-se que existem vários pontos de melhoria nos métodos experimentados.

4.2 EXPERIMENTO 2

A partir da análise de resultados do experimento anterior, foram feitos testes para verificar o impacto da métrica de similaridade/dissimilaridade a partir de análise visual dos dendrogramas gerados. O Método de Ligação Média e *Opcodes* foram escolhidos a partir dos melhores resultados obtidos, conforme apresentado na Tabela 4.8.

- **Características:** *Opcodes*
- **Método de Ligação:** Ligação Média (*Average Linkage*)
- **Métricas utilizadas:** Cosseno (*Cosine*), Jaccard e Jaro Winkler
- ***Dataset:*** Foram selecionadas subconjuntos de 10, 30 e 50 amostras do conjunto do Experimento 1 (Tabela 4.1).

O dendrograma do conjunto de 10 amostras utilizando a medida Cosseno é apresentado na Figura 4.9.

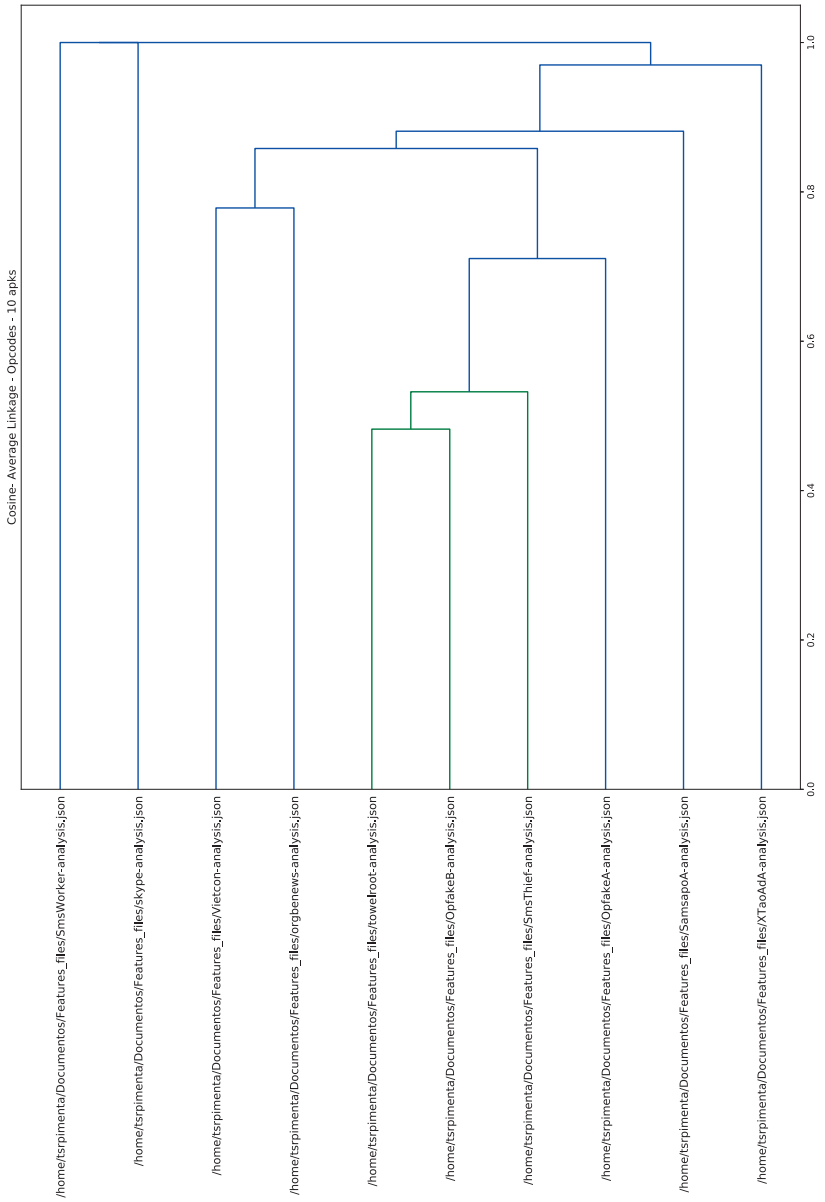


Figura 4.9: Dendrograma do conjunto de 10 amostras

Os elementos de maior similaridade entre si desse conjunto foram agrupados no *cluster* verde : OpfakeB, towelroot e SmsThief. Ao realizarmos um corte na árvore, próxima ao valor 1 (eixo vertical), verifica-se que as amostras SmsWorker e Skype infectada, no *cluster* mais a esquerda, foram as menos semelhantes em relação ao conjunto.

O dendrograma do conjunto de 10 amostras utilizando a medida Jaccard é apresentado na Figura 4.10.

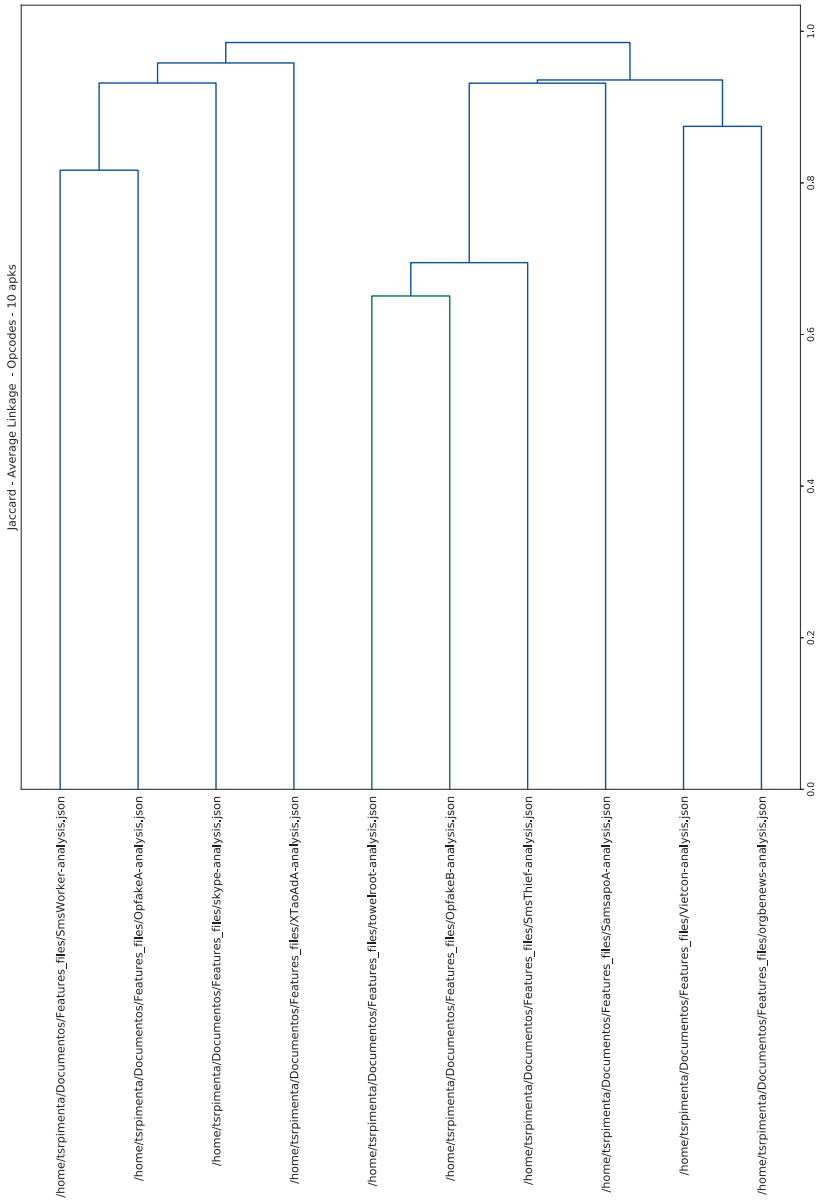


Figura 4.10: Dendrograma do conjunto de 10 amostras

Em relação à árvore resultante do uso de Jaccard para cálculo da similaridade, com uma poda perto do valor 1 (eixo vertical), pode-se visualizar que os elementos foram divididos em dois grandes grupos. Diferentemente da árvore gerada a partir do Cosseno, onde o grupo mais a esquerda teve apenas dois elementos, no caso do dendrograma da Figura 4.10, quatro amostras ficaram em um grupo e seis amostras no grupo à direita. Os elementos OpfakeB, towelroot e SmsThief novamente foram considerados os mais semelhantes entre si, porém por meio dessa métrica, a similirade entre eles foi considerada menor em relação à medida Cosseno.

O dendrograma do conjunto de 10 amostras utilizando a medida Jaro Winkler é apresentado na Figura 4.11.

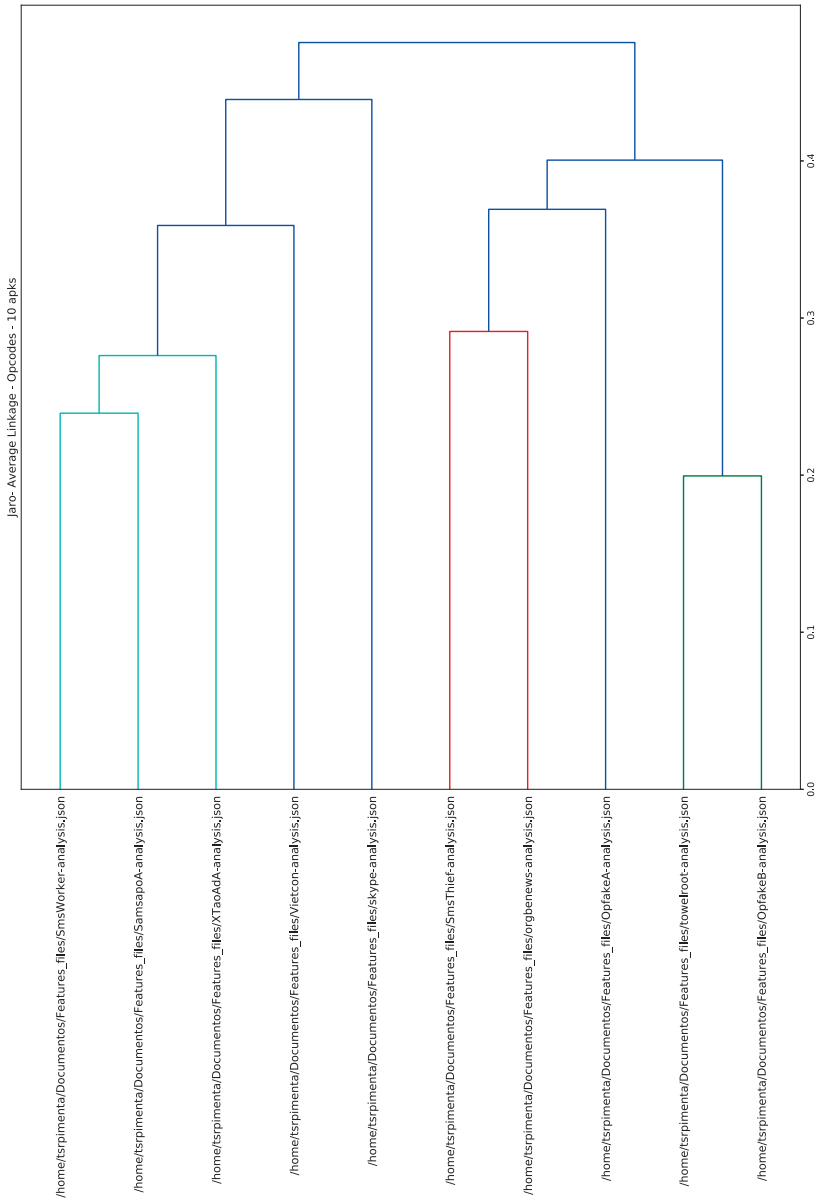


Figura 4.11: Dendrograma do conjunto de 10 amostras

A divisão dos elementos por meio da métrica de Jaro Winkler apresentou dois grandes grupos mais uniformes, em comparação aos dois dendrogramas anteriores. Os elementos OpfakeB e Towelroot novamente ficaram com o *cluster* mais similar (*cluster*verde). A amostra Skype foi considerada a menos similar ao conjunto.

O dendrograma do conjunto de 30 amostras utilizando a medida Cosseno é apresentado na Figura 4.12.

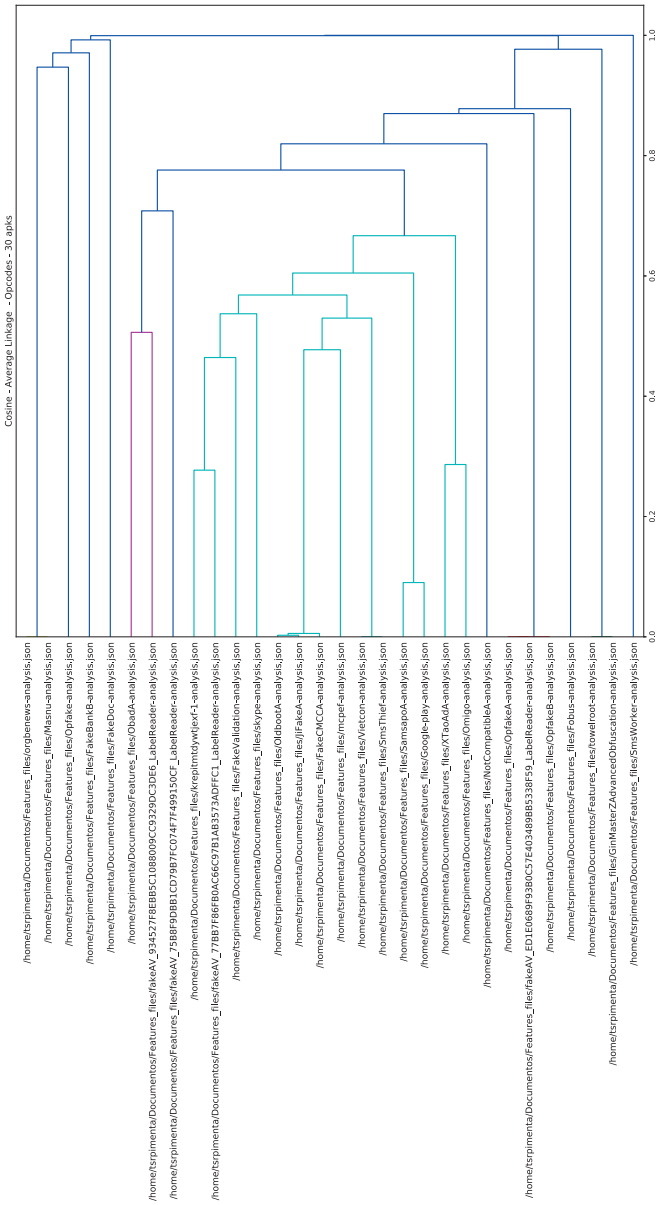


Figura 4.12: Dendrograma do conjunto de 30 amostras

Com uma poda próximo ao valor 1 (eixo vertical), pode-se visualizar dois grupos, em que o grupo mais a esquerda ficou apenas com quatro elementos. O *cluster* à direita possui as amostras com maior similaridade entre si, especialmente com as cores rosa e azul claro. Nesse conjunto, os elementos OldbootA e JiFakeA foram considerados os mais semelhantes, enquanto a amostra SmsWorker foi a menos similar ao conjunto de amostras.

O dendrograma do conjunto de 30 amostras utilizando a medida Jaccard é apresentado na Figura 4.13.

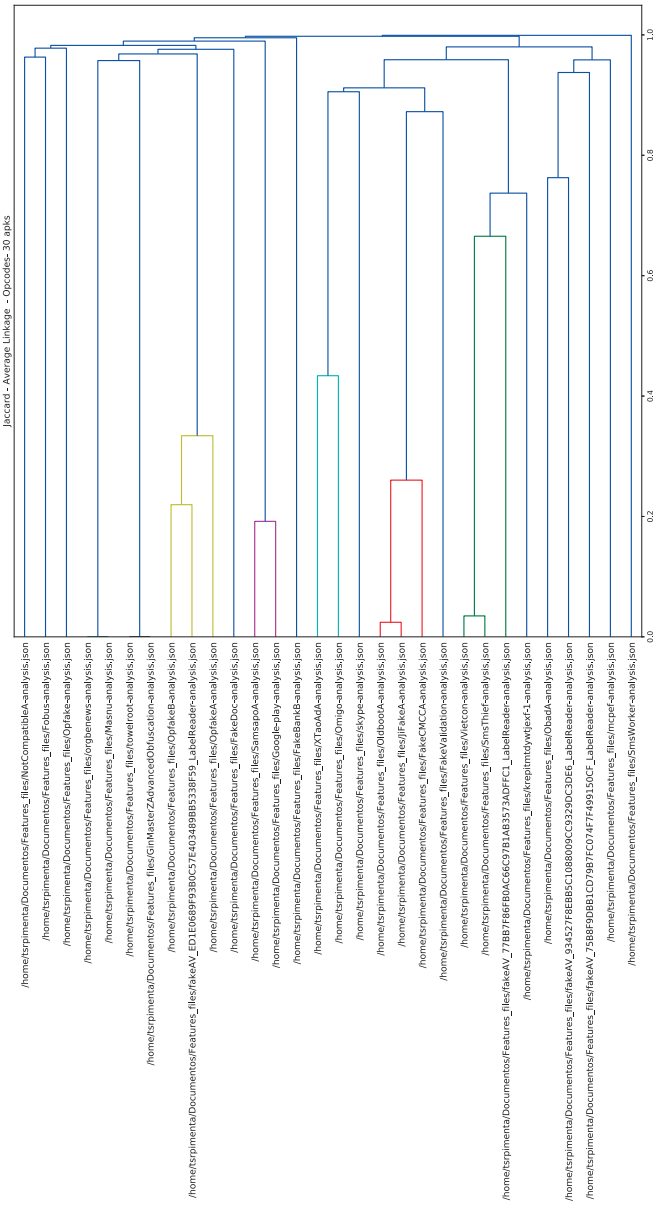


Figura 4.13: Dendrograma do conjunto de 30 amostras

Na árvore da Figura 4.13, verifica-se que os pares OldbootA e JiFakeA (*cluster* vermelho), Vietcon e SmsThief (*cluster* verde) foram os mais semelhantes do conjunto. Essas duas duplas de elementos ficaram no grupo à esquerda da árvore. No outro grande grupo (à direita), SamsapoA e Google-play (*cluster* rosa) foram os elementos considerados com maior similaridade.

O dendrograma do conjunto de 30 amostras utilizando a medida Jaro Winkler é apresentado na Figura 4.14.

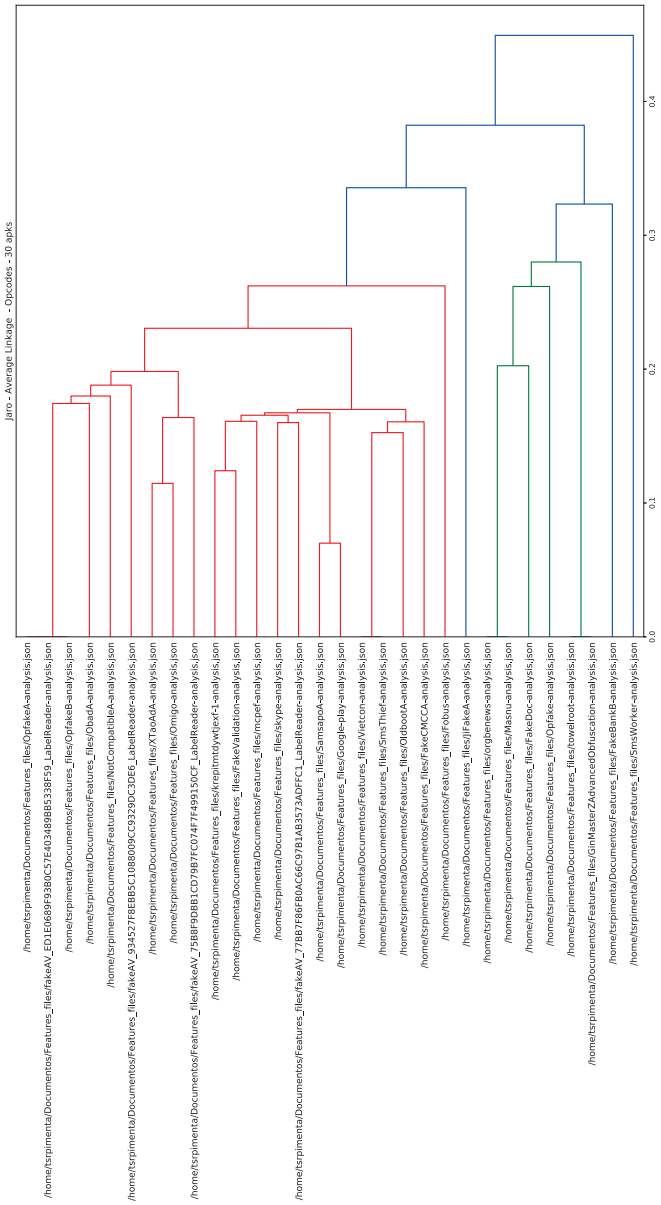


Figura 4.14: Dendrograma do conjunto de 30 amostras

Em comparação às duas árvores anteriores, a medida de Jaro Winkler considerou a similaridade total maior entre o conjunto, com o valor máximo de 0.4 na escala (eixo vertical). Foram criados dois grandes *clusters*: o vermelho à esquerda, com os elementos mais similares entre si. O verde, á direita, com elementos com pouca similaridade entre si e com o restante do conjunto. A amostra SmsWorker ficou como *outlier* do *dataset*, estando mais próxima ao grupo verde em relação aos elementos do grupo vermelho.

O dendrograma do conjunto de 50 amostras utilizando a medida Cosseno é apresentado na Figura 4.15 e o á árvore resultante com o uso da métrica de Jaccard é apresentada na Figura 4.16.

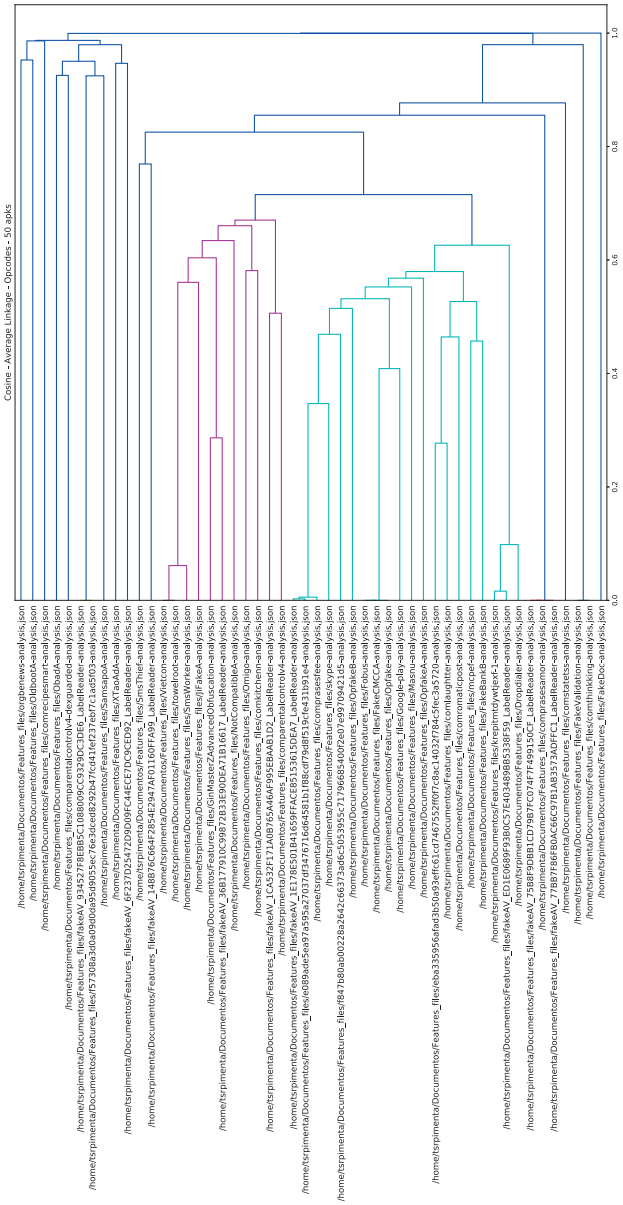


Figura 4.15: Dendrograma do conjunto de 50 amostras

Analisando visualmente as duas árvores, houve algumas semelhanças, como por exemplo, o elemento FakeDoc foi considerado como *outlier* nos dois dendrogramas. Além dessa amostra, o elemento Comthinkking, mesmo estando mais relacionado ao grupo à direita nas duas árvores, também foi considerado dissimilar ao restante do conjunto. O trio fakeAV_1E178E501B41659FFACE85153615DEA7_LabelReader, e089ade5ea97a595a27037df3476716d64581b1f88cdf79d8f519cfe431b91e4 e comprasesfee também foram avaliados com alta similaridade nos dois dendrogramas. Uma diferença entre os dois agrupamentos é a quantidade de elementos do *cluster* à esquerda, com oito elementos na árvore da Figura 4.16 e nove elementos na Figura 4.15.

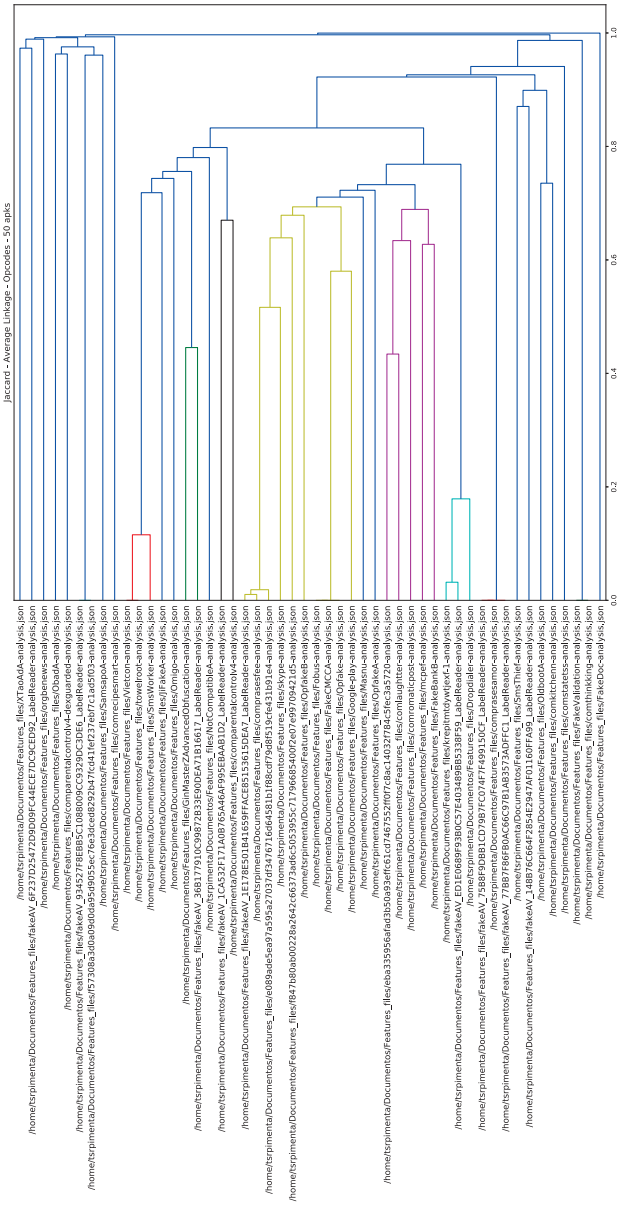


Figura 4.16: Dendrograma do conjunto de 50 amostras

Analisando as alturas dos ramos da árvore (Figura 4.16), verifica-se que os elementos mais similares entre si ficaram nos *clusters* azul claro, vermelho, verde e amarelo escuro. As amostras XTaoAdA, fakeAV_6F237D25472D9D09FC44ECE7DC9CED92_LabelReader,orgbenews, ObadA,dexguarded,fakeAV_934527F8EBB5C1088009CC9329DC3DE6_LabelReader, f57308a3d0a09d0da95d9055ec76e3dced8292b47fcd41fef237ebf7c1ad5f03, SamsapoA e comrecipesmart ficaram no grupo à esquerda, apresentando semelhança baixa entre si e alta dissimilaridade em relação ao restante do grupo.

O dendrograma do conjunto de 50 amostras utilizando a medida Jaro Winkler é apresentado na Figura 4.17.

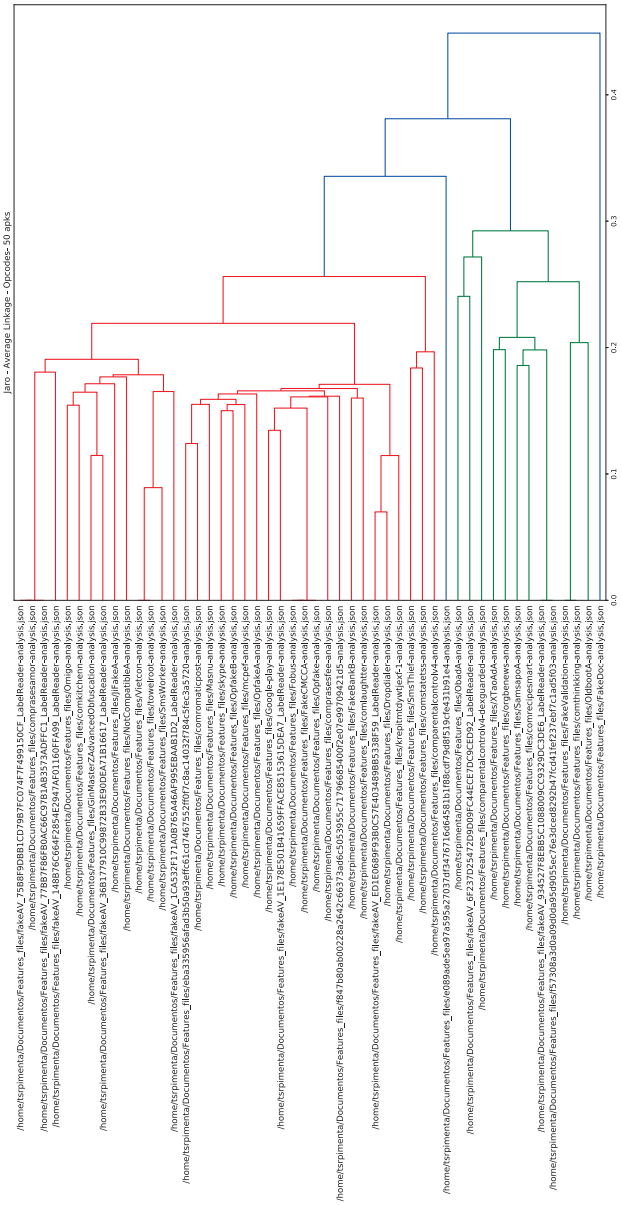


Figura 4.17: Dendrograma do conjunto de 50 amostras

Visualmente, a árvore gerada com a métrica de Jaro Winkler apresentou maiores diferenças em relação às árvores geradas a partir do Cosseno e de Jaccard. Dois grandes *clusters* foram gerados, o grupo com a cor vermelha a esquerda e o grupo verde. Analisando as alturas dos ramos verifica-se que as amostras mais semelhantes ficaram no grupo vermelho. O elemento e089ade5ea97a595a27037df3476716d64581b1f88cdf79d8f519cfe431b91e4 ficou em um ramo azul, entre os grupos vermelho e verde, considerado anômalo em relação aos demais.

A partir deste experimento, as seguintes considerações puderam ser estabelecidas:

- As medidas Cosseno e Jaccard apresentaram árvores com mais semelhanças entre si em relação à métrica de Jaro Winkler.
- O número de *clusters* gerados com as três medidas foi variável para todos os conjuntos: 10, 30 e 50 amostras.
- Elementos com grande dissimilaridade em relação ao conjunto ficaram bem separados, independente da medida utilizada.

- A representação de dados utilizada influenciou no resultado, devido às equações de cálculo de cada métrica. Como exemplo, atributos em forma de texto tiveram impacto na similaridade máxima de cada árvore, sendo melhor representadas nos agrupamentos que utilizaram a métrica de Jaro Winkler.

4.3 EXPERIMENTO 3

No presente experimento, objetivou-se avaliar o uso de exemplares não maliciosos em conjunto com *malware*. Foram analisados 111 códigos maliciosos do tipo Scareware, disponíveis no banco de dados CICAndMal2017 da UNB - University of New Brunswick (Lashkari et al., 2018). Além das amostras de *malware*, amostras de 10 aplicativos benignos (*goodware*) foram analisadas para extração de recursos e determinação de padrões. Na Tabela 4.9 são apresentadas informações sobre o conjunto de amostras analisadas. O gráfico da Figura 4.18 exibe os nomes dos aplicativos e a distribuição de amostras maliciosas e não maliciosas.

Foi utilizado o Androguard (Desnos et al., 2013) para pré-processar/descompactar os aplicativos. Os arquivos *xml* extraídos e os valores das características foram armazenados em uma base de dados para posterior análise de similaridade e agrupamento. Para a implementação, foi empregada a biblioteca `sklearn` e a linguagem Python, com o banco de dados Maria DB.

Outras alterações nesse experimento foram o vetor de características utilizado e a medida de similaridade. Foi utilizada a métrica de Levenshtein (Levenshtein, 1966) para calcular a semelhança de amostras de *malware* usando *strings* extraídas de atributos, conjunto de permissões, atividades, serviços, classes externas e receptores. A métrica de Levenshtein emprega o número mínimo de inserções, exclusões e substituições necessárias para transformar uma *string* em outra.

Tabela 4.9: Dataset - Aplicações não maliciosas e maliciosas analisadas

Nome	**Amostras**	**Malicioso ✗ ou Benigno? ✓?**
Android.spy.277	6	✗
Android Defender	17	✗
AvForAndroid	10	✗
AvPass	10	✗
FakeApp	10	✗
FakeApp AL	11	✗
FakeAV	10	✗
FakeJobOffer	9	✗
FaketaoBao	8	✗
Penetho	10	✗
VirusShield	10	✗
Among us	5	✓
Instagram	6	✓
Garena Free Fire	5	✓
Google Meet	10	✓
Messenger Text and Video	5	✓
Netflix	7	✓
Telegram	5	✓
Tik Tok	5	✓
Whats App	10	✓
Youtube	6	✓

Na escolha do algoritmo de agrupamento optou-se pelo método hierárquico aglomerativo, com ligação média (*Average Linkage*), por apresentar os resultados mais significativos no Experimento 1. Nesse método o critério é a distância de todos os indivíduos de um grupo para cada grupo. Além desse método, foram feitos testes com o algoritmo K-means.

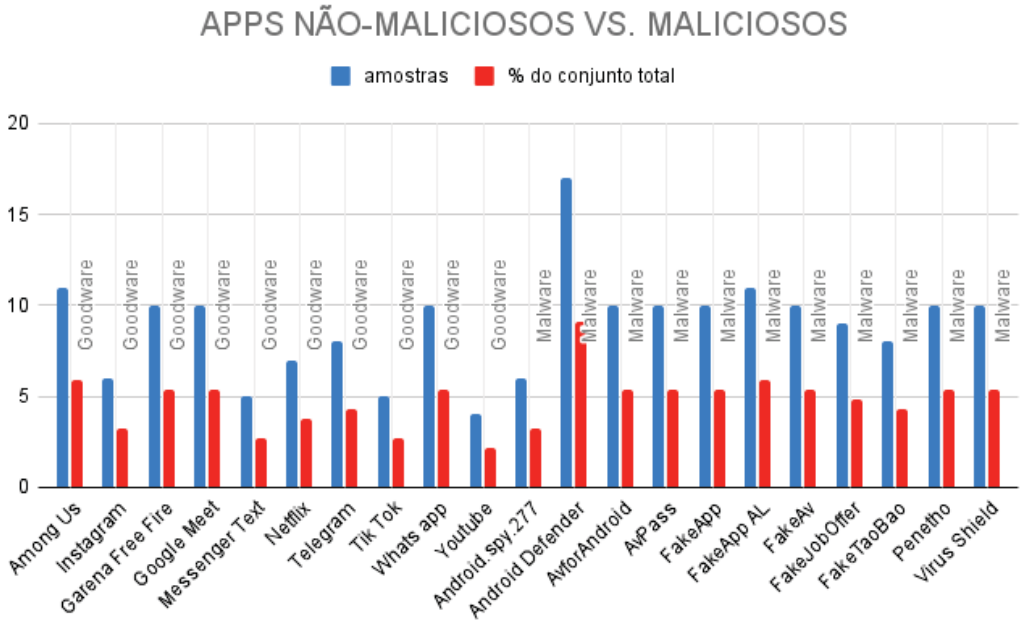


Figura 4.18: Informações sobre o conjunto de dados não malicioso e malicioso. A coluna esquerda mostra o número de amostras de cada aplicação/família. A coluna direita mostra a porcentagem do conjunto de dados total (aplicativos maliciosos e não maliciosos).

4.3.1 Resultados do agrupamento de aplicativos não maliciosos

Essa seção apresenta gráficos e dendrogramas dos grupos resultantes do agrupamento dos exemplares benignos. Em relação à escala utilizada no eixo vertical (similaridade), duas amostras idênticas têm valor 0. Consequentemente, quanto mais divergentes forem as duas amostras, maior será esse valor.

Para a execução do algoritmo K-means (Figura 4.20), as características selecionadas foram as chamadas a classes externas, permissões e atividades. Observou-se que os aplicativos Tik Tok e Among Us se destacam pela maior e menor quantidade desses atributos, respectivamente. Ressalta-se que como nem todas as aplicações possuíam o uso de serviços e receptores, esses atributos não foram empregados neste primeiro experimento.

A árvore resultante do agrupamento hierárquico das amostras de *goodware* é mostrada na Figura 4.19. Notou-se que a grande maioria das aplicações foram devidamente agrupadas. No entanto, exemplares do Google Meet foram divididos em dois grupos separados (*clusters* G e H). Essa separação justifica-se porque as amostras do Grupo H possuem menor quantidade de todos os atributos quando comparadas às amostras mais recentes, colocadas no outro grupo.

De acordo com a análise da similaridade dos atributos exclusivos ou similares, percebeu-se que aplicativos em um mesmo *cluster* possuíam dezenas de recursos idênticos e, em grupos separados, os aplicativos possuíam centenas de recursos diferentes. Por exemplo, as amostras de aplicativos do Instagram e do Whats App tinham 24 permissões, três serviços, nove classes externas e duas atividades em comum. Comparando os conjuntos de características de amostras

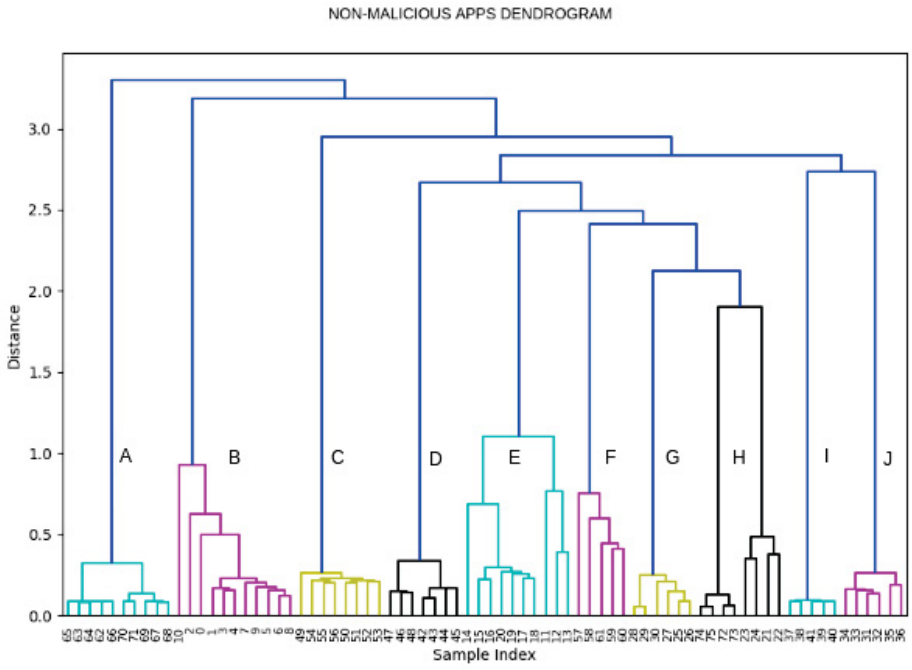


Figura 4.19: Dendrograma da classificação das amostras de *goodware*

puderam

do Instagram e do Tik Tok, foram detectadas diferenças em 355 atividades, 1238 classes, 66 serviços, 32 receptores e 53 permissões.

1. **Whats App - *Cluster* A**. Como diferenciais em relação aos outros aplicativos de comunicação analisados, as amostras do Whats App possuem permissões para receber mapas, serviço de backup, NFC, instalação de pacotes, leitura de números de telefone, visualização e alteração de configurações. Comparando os elementos de índices 62 e 66, a diferença entre os exemplares foi de 24 atividades, 275 classes externas, três serviços, e uma permissão. A permissão adicionada nas cinco amostras mais recentes foi a `android.permission.READ_PHONE_NUMBERS`. Apenas estudando o total das características de cada versão, não seria provável considerar essa dissimilaridade. Também verificou-se que houve diversas diferenças nos tamanhos dos *apks* do conjunto de amostras desse aplicativo.

2. **Jogo Among Us - *Cluster* B**. Ao contrário do Garena Free Fire, esse jogo não possui múltiplos atributos com segurança questionável. O exemplar de versão v2020.6.9 é um *outlier* do conjunto. Houve algumas alterações no tamanho das amostras.

3. **Telegram - *Cluster* C.** As amostras 37 e 38 foram as mais semelhantes no conjunto do Telegram. Ao comparar as amostras 32 e 39 foram encontradas diferenças em 10 classes, um serviço e um receptor. O conjunto de permissões e atividades foi 100% herdado da versão mais antiga. Sobre o tamanho, houve um acréscimo nos elementos mais recentes.

4. **Netflix - *Cluster* D.** Ao verificar a quantidade total de atributos de cada exemplar, a diferença entre eles parece sucinta, porém, as atividades em comum com a versão mais

antiga do conjunto, variaram cerca de 50,5%, enquanto as classes externas atingiram 59% de variação. O tamanho do arquivo teve um acréscimo nos elementos mais recentes.

5. **Garena Free Fire - *Cluster* E.** O total de classes externas utilizado pelo jogo é expressivo. Uma questão de segurança notória é: qual o interesse desse jogo em obter informações do dispositivo sobre as tarefas em execução ou executadas recentemente? Além disso, esse jogo possui outras permissões consideradas perigosas, como gravação de áudio, obtenção do status da rede e do Bluetooth. Examinando o total de atributos, fica evidente uma divisão dos grupos por conta das dissimilaridades dos números totais de permissões, receptores e serviços. Por exemplo, as alterações detectadas entre as amostras 51 e 60 foram de três permissões, oito atividades, três receptores, sete serviços e 62 aulas.

6. **Tik Tok - *Cluster* F.** Algumas permissões diferenciadas são a leitura de mídia de armazenamento externo e de acesso ao estado das conexões Wifi. Diversos outros pacotes são usados para ler e modificar as configurações gerais do dispositivo. As diferenças nas amostras do Tik Tok são perceptíveis nas quantidades de recursos: um incremento no número de classes, de receptores e de atividades. Essas alterações dividiram as cinco versões em dois grupos. Ao comparar as amostras 32 e 39, detectamos diferenças de um receptor, cinco serviços, 28 atividades e de 115 classes. O tamanho teve um acréscimo nos exemplares mais recentes.

7. **Google Meet - *Cluster* G e *Cluster* H.** Em comparação com a versão mais antiga, houve divergências entre as classes externas chamadas, variando entre 25 e 117 classes distintas. Comparando os elementos 22 e 31, por exemplo, foram verificadas diferenças em 12 serviços, 117 classes, 19 receptores, 38 atividades e 17 permissões. Além disso, houve algumas flutuações nos tamanhos das amostras.

8. **Youtube - *Cluster* H.** Os exemplos do YouTube diferem devido ao crescimento do número de classes e diminuição de atividades nas amostras mais recentes. Os elementos com índices 6 e 7 tiveram diferenças de 217 classes, tornand*hash* 00001ac7364e668f1ddc9906887edf0c8f7230830b864db6179075ac9c0a689o esses elementos como o par menos semelhante do grupo. O conjunto de permissões, serviços e receptores foi 100% herdado da versão YouTube.v16.01.34. Para os conjuntos de atividades e classes utilizadas, apesar das mudanças na quantidade total, a herança da versão mais antiga do conjunto foi superior a 83%. Em relação ao tamanho, houve redução nos elementos mais recentes.

9. **Messenger - *Cluster* I.**Esta é a aplicação de comunicação com o maior número de atributos. Os elementos do Messenger possuem várias permissões do Facebook como escrita e recebimento de mensagem SMS e ligação telefônica. As cinco versões do Messenger foram separadas pelo total de classes externas utilizadas. O conjunto de permissões e receptores foi 100% herdado da versão mais antiga. No entanto, ao comparar as amostras 71 e 75, as discrepâncias encontradas foram de três atividades, de 607 aulas e de um serviço. Em relação ao tamanho, houve um aumento nos elementos mais recentes.

10. **Instagram - *Cluster* J.** A separação das seis versões do Instagram pode ser feita com base na adição do número de serviços e receptores e na diminuição do número de classes nas amostras 4 e 5. A permissão adicionada no Instagram.v174.0 foi

a `android .permission.READ_PHONE_NUMBERS`, enquanto a atividade adicionada foi `com.instagram.rtc.activity.RtcGridSandboxActivity`. Houve várias alterações no número de classes externas chamadas nos exemplares. Os três serviços adicionados nas duas últimas versões estavam relacionados a atividades executadas em segundo plano e ao sistema de alarme. O tamanho teve acréscimo nos elementos mais recentes, exceto no último.

Validação dos resultados do agrupamento Para os 10 *clusters* gerados, o valor de Precisão foi igual a 1. Os valores de Recall e F1-Score foram iguais a 1 em quase todos os *clusters*, exceto no *cluster* H onde Recall foi 0,6 e F1-Score foi 0,75.

4.3.2 Resultados do agrupamento de *malware*

O resultado do agrupamento de famílias de *malware* usando K-means é mostrado na Figura 4.20. Devido a limitações de espaço, a lista de amostras analisadas, o número de feições e árvores adicionais estão no Apêndice. Percebeu-se que houve confusão no agrupamento das amostras, tanto nos resultados K-means quanto nos grupos resultantes do agrupamento hierárquico (HAC). Além disso, informações sobre atributos únicos de cada família estão na Tabela 4.10.

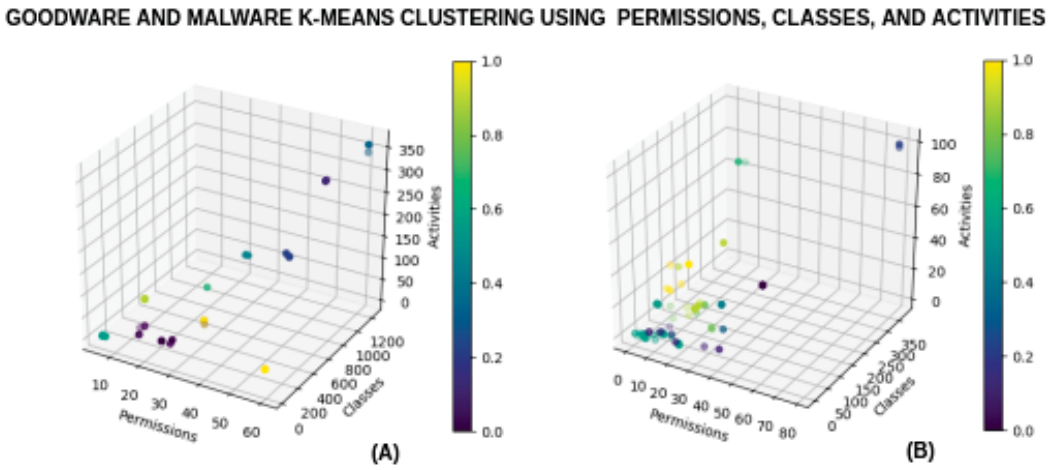


Figura 4.20: Resultados do *cluster* K-Means Goodware(A) e *malware* (B) usando atividades, classes e permissões.

(A) **Android spy 277**. As permissões mais perigosas são acesso à câmera, gravação de áudio, mensagens SMS e autorização para desativar o bloqueio de teclas e de qualquer segurança de senha associada. No agrupamento de famílias de *malware*, as amostras dessa família foram separadas em *clusters* roxo e azul claro. Analisando o total de recursos, as amostras 3 e 4 seriam as mais semelhantes entre si se comparadas ao par 4-5. Enquanto isso, as amostras 3 e 4 têm apenas uma atividade e dez permissões em comum, enquanto o par 4-5 compartilha uma atividade, sete classes e dez permissões. Ressaltamos que houve várias mudanças no tamanho das amostras.

(B) **Android Defender.** Essa família contém o maior número de amostras e características exclusivas (Tabela 4.10). Após agrupar apenas as amostras dessa família, notou-se que três grupos foram formados. Os pares de elementos mais semelhantes foram 15-17, 8-20 e 9-10. No agrupamento de todas as amostras de *malware*, quatro amostras dessa família foram separadas em um grupo, demonstrando que são distintas das demais desse

Tabela 4.10: Características exclusivas de cada família.

Família	Permissões	Atividades	Receptores	Classes	Serviços
Android.spy.277	1	14	4	47	2
Android Defender	14	22	9	165	13
AvForAndroid	0	43	9	51	4
AvPass	0	13	2	74	2
FakeApp	0	1	0	1	0
FakeApp AL	2	1	1	26	1
FakeAV	4	5	4	6	7
FakeJobOffer	1	24	4	8	3
FaketaoBao	0	1	4	0	2
Penetho	0	3	0	2	0
Virus Shield	3	7	12	59	15

conjunto. Notou-se que esse tipo de *malware* utiliza diversos serviços e classes externas com nomes distintos, principalmente para leitura e modificação de configurações do sistema. Essa família contém amostras com quantidades heterogêneas de atributos e de tamanhos.

(C) **AvForAndroid.** Essa família se destacou com o maior número de atividades exclusivas (Tabela 4.10). No agrupamento, oito elementos estavam em um agrupamento e dois em outro grupo. É interessante comparar amostras de uma mesma família para verificar comportamentos diferentes. Por exemplo, o exemplo 23 tem permissões para alterar o estado da rede, gravar áudio e outras 19 permissões distintas em relação ao grupo. Além disso, o exemplo 25 pode instalar atalhos, obter contas de usuário e obter a lista de sites favoritos.

(D) **AVPass.** Essa família acessa várias classes externas exclusivas (Tabela 4.10). Possui inúmeras atividades relacionadas a livros e algumas relacionadas a ações de *sniffer*. Alguns exemplos notáveis são espionagem em atividades de compras, como carrinho de compras, *zoom* em produtos e também em promoções. Todas as amostras estavam no mesmo *cluster*. Para as amostras semelhantes de índices 38, 36 e 39, detectou-se que o que separava o elemento 38 do par 36-39 eram mudanças nas classes e atividades.

(E) **FakeApp.** Esse grupo tem o conjunto mais simples de atributos exclusivos. Algumas das permissões que se destacam são `android.permission.KILL_BACKGROUND_PROCESSES`, `com.android.browser.permission.WRITE_HISTORY_BOOKMARKS`, e `android.permission.BLUETOOTH_ADMIN`.Analisando os elementos dessa família, percebe-se que a amostra 47 foi uma anomalia (*outlier*, principalmente pelo aumento do número de permissões. Também houve várias mudanças no tamanho das amostras desse conjunto.

(F) **FakeApp AL.** Essa família possui o maior conjunto de classes exclusivas (Tabela 4.10). As amostras foram divididas em dois *clusters*. No agrupamento, os pares mais semelhantes foram 54-59 e 60-61. Permissões incomuns encontradas são o de status da bateria, acesso ao compartilhamento Bluetooth, ligar a lanterna do dispositivo, a câmera e executar chamadas telefônicas.

Tabela 4.11: *Clusters* resultantes do agrupamento hierárquico das amostras maliciosas

Cluster	Amostras
Cluster 1	4 Android Defender
Cluster 2	5 Android Spy277, 13 Android Defender, 8 AvForAndroid, 10 AvPass, 10 fakeApp, 6 fakeAppAL, 10 fakeAV, 8 faketaoBao, 9 Virus Shield, 10 Penetho
Cluster 3	11 Android Defender, and 9 fakeJobOffer
Cluster 4	5 FakeApp AL
Cluster 5	2 AvForAndroid, 1 Virus Shield, and 1 Android Spy277

Tabela 4.12: Métricas calculadas (%) para resultados do agrupamento de *malware* (cada *cluster* é indicado por C seguido de seu número).

Métrica/*cluster*	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
Precisão	100	90	89	64	70	45	80	0	37	100	0
Recall	100	100	100	100	100	100	57	0	75	25	0
F1-Score	100	94	94	78	82	62	66	0	50	40	0

(G) **FakeAV.** O elemento 70 foi um *outlier* nesse conjunto devido ao número de classes. No entanto, todas as amostras permaneceram no mesmo grupo. As permissões mais perigosas encontradas nesse *malware* foram `android.permission.CLEAR_APP_CACHE` , `android.permission.INJECT_EVENTS`, `android.permission.DELETE_PACKAGE` `android.permission.INSTALL_PACKAGES`.

(H) **FakeJobOffer.** Essas amostras foram agrupadas com elementos do conjunto Android Defender. Simplesmente pelo número de atributos, pode-se considerar todas as amostras equivalentes. No entanto, os pares mais similares foram 86-88 e 90-91. A permissão exclusiva encontrada nessa família foi a `com.saavn.android.permission.C2D_MESSAGE`.

(I) **Faketaobao.** Essa família apresenta poucos receptores, serviços e atividades exclusivas. Algumas funcionalidades que se destacam estão relacionados à transmissão de dados da inicialização do dispositivo, recebimento de SMS e atividades executadas em segundo plano. Embora as amostras estejam todas no mesmo *cluster*, os elementos mais divergentes são os de índices 83 e 87 e os mais semelhantes são os pares 84-89 e 88-90.

(J) **Penetho.** Apesar de estar no mesmo *cluster*, o elemento 92 foi separado pela diferença no número de permissões, atividades e classes. É interessante notar que essas amostras não utilizam receptores e serviços.

(K) **Virus Shield.** Essa família tem o conjunto mais significativo de receptores exclusivos (Tabela 4.10). O elemento 107 foi um *outlier* nesse conjunto devido ao número de classes, sendo, portanto, agrupado no mesmo *cluster*. O par 103-108 foi o menos dissimilar do conjunto. O número total de características e o tamanho de cada amostra foram heterogêneos.

Validação do *cluster*. Conforme mostrado na Tabela 4.12, os valores da validação do agrupamento foram variados, sendo os valores de Precisão mais baixos nos *clusters* 6, 9 e 11.

Tabela 4.13: *Clusters* e as amostras de *Goodware* e *Malware*.

Aplicação	***Cluster***	**Nº de amostras**
Among us	Cluster 10	todas
Garena Free Fire	Cluster 11	todas
Google Meet	Cluster 14	6
	Cluster 16	4
Instagram	Cluster 7	todas
Messenger	Cluster 6	todas
Netflix	Cluster 8	todas
Telegram	Cluster 5	todas
Tik Tok	Cluster 13	todas
Youtube	Cluster 15	todas
Whats App	Cluster 1	todas
Android.spy.277	Cluster 19	1
	Cluster 22	4
	Cluster 26	1
Android Defender	Cluster 9	10
	Cluster 12	3
	Cluster 29	1
	Cluster 22	3
AvForAndroid	Cluster 19	2
	Cluster 22	4
	Cluster 26	3
	Cluster 28	1
AvPass	Cluster 18	7
	Cluster 20	2
	Cluster 26	1
FakeApp	Cluster 22	1
	Cluster 27	5
	Cluster 28	1
	Cluster 29	3
FakeApp AL	Cluster 3	1
	Cluster 17	5
	Cluster 19	2
	Cluster 24	1
	Cluster 28	2
FakeAV	Cluster 10	9
	Cluster 21	1
FakeJobOffer	Cluster 5	8
	Cluster 21	1
FaketaoBao	Cluster 22	1
	Cluster 23	4
	Cluster 25	2
	Cluster 29	1
Penetho	Cluster 2	todas
Virus Shield	Cluster 19	1
	Cluster 21	3
	Cluster 22	2
	Cluster 24	3
	Cluster 25	1

4.3.3 Agrupamento de amostras de não maliciosas e maliciosas

O agrupamento K-means de todas as amostras (conjunto de dados de *malware* + goodware) é mostrado na Figura 4.21. Devido ao tamanho do dendrograma, a figura pode ser encontrada no Apêndice.

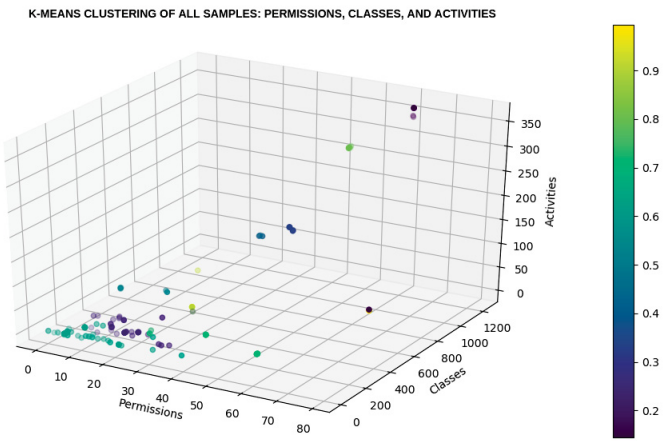


Figura 4.21: Agrupamento K-means de todas as amostras.

A maioria dos aplicativos benignos foi agrupada corretamente, exceto por algumas amostras do aplicativo Google Meet. Sobre as amostras maliciosas, a classificação da família Penetho se destaca com todas as amostras no mesmo grupo. Observou-se que ao comparar os atributos similares entre as amostras do jogo Garena Free Fire e do *malware* FakeApp_AL, foram encontradas treze permissões inseguras em comum. Essas permissões já foram mencionadas anteriormente nas análises desses grupos. Infelizmente, os conjuntos de atributos escolhidos e a medida de similaridade não foram satisfatórios para separar as famílias, mas foram capazes de distinguir entre amostras maliciosas e não maliciosas.

4.3.4 Validação do Agrupamento

A tabela para validação dos *clusters* de todas as amostras pode ser encontrada no Apêndice. Observou-se que além dos *clusters* com valores 0, os menores valores de precisão, acurácia e F1-score foram os dos grupos 20,22. Os índices adotados para examinar os resultados dos *clusters* são mostrados na Tabela 4.14. De acordo com os resultados, ficou claro que o agrupamento de amostras não maliciosas teve os melhores resultados, chegando a 95% de acerto. Em relação ao agrupamento de famílias de amostras maliciosas e do conjunto total de dados (*malware* e goodware), algumas medidas revelaram que este último teve melhor desempenho. De acordo com os resultados da validação do agrupamento de amostras maliciosas, verificou-se que o recall não foi satisfatório, indicando que embora existam características exclusivas, o conjunto de atributos iguais também é grande. Isso faz com que as amostras sejam pouco distinguíveis, criando uma mistura entre os grupos. Embora as amostras não maliciosas sejam mais distinguíveis, a classificação das maliciosas acabou reduzindo a precisão do agrupamento de todas ás amostras do conjunto de dados.

4.4 EXPERIMENTO 4

A presente seção apresenta informações sobre os exemplares analisados, métodos de extração de características e de agrupamento aplicados no terceiro experimento (Figura 4.22).

Tabela 4.14: Índices calculados (%) para os resultados de agrupamento de goodware, *malware* e goodware-*malware*

Agrupamento	**Acurácia**	**Precisão**	**Recall**	**F1-score**	**Homogeneidade**	**V-Measure**
Goodware	95	1	94	96	100	98
malware	64	83	64	63	72	66
Goodware-*malware*	67	80	67	71	89	85

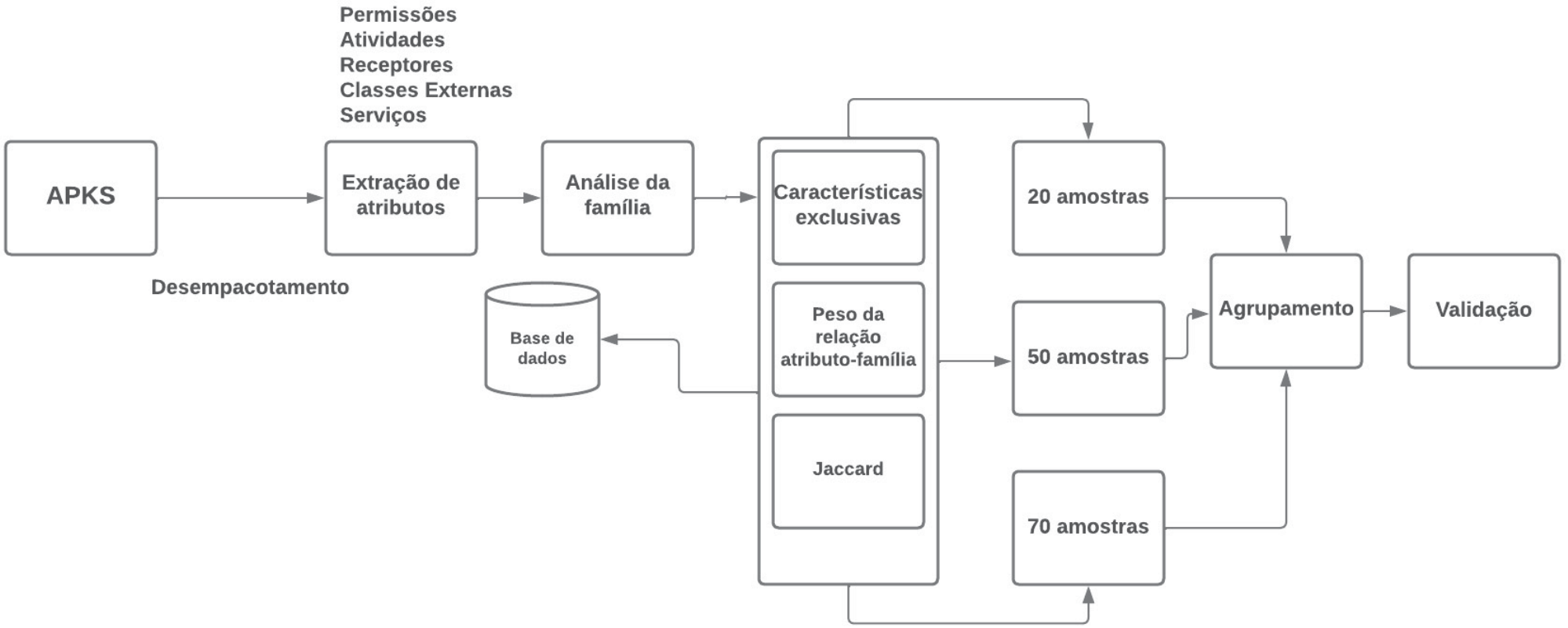


Figura 4.22: Método aplicado

Para as análises foram examinados aplicativos disponíveis na base Android Botnet 2015, da UNB - University of New Brunswick (A. F. A. Kadir e Ghorbani, 2015). Informações sobre as famílias dos exemplares estudados são apresentadas na Tabela 4.15. Foram selecionados, aleatoriamente, 70 exemplares de cada família para análise de similaridade.

Como pré-processamento dos aplicativos, para o desempacotamento dos arquivos *.apks* utilizou-se o Androguard (Desnos et al., 2013). As características extraídas foram armazenadas em uma base de dados, para análise de similaridade e agrupamento. Acerca da comparação de similaridade entre as amostras de *malware*, optou-se pelo índice de Jaccard (Cha, 2007). Foram feitos agrupamentos com os valores da similaridade do conjunto dos atributos (permissões, atividades, serviços, classes externas e receptores) e para cada atributo isolado.

Posteriormente, após o estudo de padrões de características específicas para cada família, foram calculados e definidos pesos para os atributos de maior significância para cada família. Além da similaridade, o cálculo dos pesos utilizou a divisão da contagem dos atributos encontrados na amostra pela quantidade total presente na família. Como exemplo, verificou-se que os serviços `"com.admv6.service.AdvService"` e `"com.admv6.service.MainService"` eram mais importantes e frequentes na família MisoSMS, enquanto os serviços `"com.android.security..SecurityService"` e `"com.systemsecurity6.gms..MainService"` apresentavam frequências e importâncias significativas na família Zitmo. Desse modo, foram definidas dezenas de relações atributo-família para os exemplares das famílias analisadas.

Em relação ao agrupamento, foram aplicados os algoritmos de Ligação Média (*Average Linkage*), Ligação Completa (*Complete Linkage*) e Ligação Simples (*Single Linkage*). Além dos métodos de classificação hierárquica, o método de agrupamento foi adaptado utilizando o conceito de medóide. Medóides são objetos representativos de um conjunto, cuja soma de dissimilaridades para todos os objetos no *cluster* é mínima (Struyf et al., 1997). Vantagens desse método, como mencionados na Seção 2.8 sobre métodos particionais, incluem rapidez, facilidade de implementação, convergência e auxílio para interpretação dos objetos de cada *cluster*. Desse modo, para o aprimoramento do método de classificação, foram calculados os medóides de cada família.

Tabela 4.15: Android Botnet 2015 Dataset - Total de exemplares e informações sobre as famílias analisadas

Família	**Ano**	**Total**	**Comportamentos**
Zitmo	2010	80	Monitora as mensagens SMS a procura de códigos usados para completar transações bancárias (S. Josh e Joshi, 2015).
DroidDream	2011	363	Utiliza meios maliciosos para obter privilégios de administrador, acessar informações pessoais presente no dispositivo, depois as envia a um servidor remoto (Y. Kim e Chan, 2016).
PJapps	2011	244	Envia mensagens SMS, rouba dados bancários, instala novas aplicações e abre URls no navegador do dispositivo (S. Josh e Joshi, 2015)
NickySpy	2011	199	Grava áudios de chamadas e os armazena em SD Card (Pieterse e Burke, 2015b)]
AnserverBot	2011	244	Primeira família de *malware* Android que lê postagens de blogs e as interpretam como comandos. (A. F. A. Kadir e Ghorbani, 2015)

TigerBot	2012	96	Rouba informações como localização GPS, encerra processos rodando, lê e exclui mensagens SMS maliciosas (S. Josh e Joshi, 2015)
MisoSMS	2013	100	Rouba mensagens SMS e as transmite, clonagem de telefone ou golpes de fraude (Pieterse e Burke, 2015b)
Wroba	2014	100	Ataques bancários baseados em localização geográfica. Rouba e compartilha informações para lista de contatos (A. F. A. Kadir e Ghorbani, 2015).
Pletor	2014	85	Comportamentos de *Ransomware*. Baixa e instala falsas atualizações de aplicativos bancários (A. F. A. Kadir e Ghorbani, 2015).
Not Compatible	2014	76	Conhecido por ser utilizado para e-mails de spam e comprometer redes inteiras (Pieterse e Burke, 2015b)

4.5 RESULTADOS E DISCUSSÃO

Nessa seção, as informações obtidas da análise de similaridade e agrupamento dos exemplares do Experimento 3 são apresentadas.

4.5.1 Permissões

A similaridade entre as permissões de cada família pode ser visualizada na Figura 4.24 (coluna amarela). Devido à grande similaridade do conjunto de permissões, as famílias Anserver e Tigerbot e as famílias Droiddream e PJapps podem ser facilmente confundidas.

As permissões mais utilizadas, em geral, pelos aplicativos maliciosos deste *dataset* relacionam-se aos seguintes eventos: SMS (enviar, ler, escrever, receber), Internet e configuração de rede, *boot*, ler dispositivo de memória externa e lista de contatos (leitura). Permissões adicionais, que evidenciam comportamentos maliciosos, são listadas a seguir:

- Status da bateria: permissão encontrada em amostras da família Anserver, MisoSMS e PJapps. Uma das limitações dos ataques de *botnets* movéis é a quantidade finita de bateria do dispositivo. Além disso, consumo alto de energia também pode indicar presença de *malware*.

- *Bookmarks:* Apesar de ser encontradas em aplicativos benignos, permissões de escrita e de consulta a histórico de *bookmarks* foram encontradas em exemplares da família PJapps. Além de roubo de dados de preferências do usuário, podem ser utilizadas para *spam* de publicidade.

- Gravação de áudio e *Upload* de arquivo de áudio: Além de gravar chamadas telefônicas, amostras da família Nickyspy apresentaram permissões que possibilitam o envio de áudios do dispositivo a servidores.

- Câmera: Permissão encontrada em amostras das famílias Nickyspy e Pletor. *Malware* da família Nickyspy enviam uma imagem da vítima, capturada via câmera frontal do *smartphone*, em golpes envolvendo dados bancários e monetários (A. F. A. Kadir e Ghorbani, 2018).

4.5.2 Serviços

A similaridade entre os serviços de cada família é apresentada na Figura 4.24 (coluna esquerda). Conforme pode ser verificado, apenas as famílias Tigerbot, Pletor, Wroba e Zitmo poderiam ser diferenciadas das demais. Entretanto, a diferença entre essas quatro famílias ainda é pouco significativa.

Alguns registros de serviços preocupantes, encontrados em famílias analisadas, são listados a seguir:

- Instalação e remoção de pacotes: Possibilita a remoção de aplicativos anti-*malware* ou a inclusão de outros programas maliciosos. Foram detectados registros dessas opções em amostras analisadas das famílias Wroba e Droiddream.

- Histórico de chamadas do dispositivo: Aplicações espiãs armazenam dados de chamadas telefônicas e os encaminham para outros usuários. Exemplares da família Tigerbot apresentaram essa funcionalidade.

- *Upload* de mensagens do dispositivo: Esse serviço pode ser utilizado para interceptação de mensagens entre aplicações bancárias e o dispositivo do usuário. Registros encontrados nos exemplares da família Tigerbot.

- Envio de SMS para outros serviços: Além de enviar mensagens custeadas pelo usuário, o *malware* pode distribuir credenciais coletadas para outros aplicativos maliciosos. Serviço encontrado em exemplares da família Wroba.

4.5.3 Receptores

A similaridade entre os receptores de cada família pode ser visualizada na Figura 4.24 (coluna direita). Assim como verificado com os serviços, apenas as famílias Tigerbot, Pletor, Wroba e Zitmo apresentaram diferença de similaridade (pouco significativa) em relação às demais.

Os principais registros encontrados no conjunto de exemplares são: `"startup receiver"`, `"on boot receiver"`, `"internet receiver"` e `"sms receiver"`. Outros receptores suspeitos são listados a seguir:

- Uso de *payloads* maliciosos: AnserverBot utiliza receptores para pedir que os usuários instalem *payloads* através de anúncios de atualizações falsas de aplicativos (Zhou e Jiang, 2011). *Payloads* são códigos que rodam silenciosamente em segundo plano.

- Receptores acionados por eventos: algumas famílias de *malware* ativam atividades quando eventos são disparados, como inicialização do dispositivo e conexão à Internet. Registros foram encontrados em exemplares da família MisoSMS.

- Gerenciamento do sistema de arquivos de cartão de memória: Além de arquivos e informações presentes no sistema de arquivos do dispositivo, é interessante para os aplicativos maliciosos examinar cartões de memória. Registros foram encontrados em amostras da família Pletor.

4.5.4 Atividades

A Figura 4.23 apresenta o gráfico de similaridade entre as atividades de cada família (coluna esquerda). O conjunto de atividades foi semelhante entre várias famílias, como nos conjuntos Droiddream-PJapps-Nickyspy e Tigerbot-MisoSMS-Pletor-Zitmo-Notcompatible.

A partir de métodos inicializados através de atividades, códigos maliciosos conseguem informações sensíveis. Por exemplo, pode-se obter informações como localização GPS (família Wroba)e credenciais de sistemas de pagamento (Paypal). Entre as atividades mais utilizadas pelas amostras maliciosas analisadas, verificam-se:

- Conteúdo pornográfico: Redirecionamento para páginas com conteúdos suspeitos podem acarretar a instalação de *malware* adicionais. Atividades com URLs suspeitas foram encontradas nas famílias PJapps e Droiddream.
- Chamadas recebidas: Em amostras da família Zitmo, foram encontradas atividades relacionadas ao registro de chamadas recebidas pelo dispositivo. Alguns *malware* utilizam gravação de áudios para chantagem de usuários. Em amostras da família Tigerbot foi possível encontrar métodos para atender chamadas automaticamente.
- *Download* de Serviços e atualizações de Segurança: Acessos a endereços de páginas *web* com atualizações falsas são armadilhas utilizadas para roubo de informações pessoais.

4.5.5 Classes

A Figura 4.23 também apresenta o gráfico da relação de classes registradas no banco de dados por família de *botnet*. Conforme pode ser visualizado no gráfico, as famílias com conjuntos de classes mais similares foram Anserver, Droiddream, Tigerbot , Pletor e Zitmo.

As seguintes funcionalidades, ativadas por classes externas, foram observadas:

- Conexão à rede Tor: Como os *botnets* necessitam de comunicação com servidores, algumas classes podem ser chamadas para acesso à rede anônima Tor (Syverson et al., 2004). Registros foram encontrados na família Pletor.
- Criptografia de arquivos: *Botnets*, com comportamentos de *ransomware*, utilizam diversas classes externas de criptografia de arquivos e de transferência de arquivos. Além de amostras da família Pletor, chamadas a esses tipos de classes aparecem em exemplares da família Anserver.
- Verificação de usuários do dispositivo: Para ultrapassar mecanismos de segurança, códigos maliciosos tentam obter privilégios de administrador do sistema. Registros desse estilo foram encontrados na família Zitmo.

4.5.6 Agrupamento e Análise de Similaridade

Para o agrupamento, foram utilizados os algoritmos de Ligação Média (*Average Linkage*), Ligação Completa (*Complete Linkage*) e Ligação Simples (*Single Linkage*). Nos primeiros experimentos, aplicaram-se os métodos de agrupamento hierárquico diretamente aos valores de similaridade das características extraídas. Os resultados de validação dos *clusters* desses experimentos não foram satisfatórios, tendo valores de precisão inferiores a 50%, utilizando 20 amostras por família. Detectou-se que houve confusão entre variantes da família Droiddream sendo classificados como da família MisoSMS, por exemplo.

Após esses experimentos, foi aplicado o agrupamento de conjuntos de atributos isolados. Por exemplo, utilizando apenas permissões ou apenas atividades para diferenciar as famílias. Os resultados da validação dos agrupamentos demonstraram que essa alternativa também não foi satisfatória. Isso deve-se à semelhança significativa dos conjuntos compartilhados entre várias famílias, conforme pode ser analisado nos gráficos das Figuras 4.24 e 4.23 . Por exemplo, o conjunto de serviços e receptores das famílias Anserver, Droiddream, PJapps, MisoSMS e NotCompatible apresentaram similaridade máxima, impossibilitando a distinção entre os exemplares dessas famílias somente com o uso dessas informações. A partir das características exclusivas de cada família, foi possível estimar o peso para cada relação atributo-família no cálculo de similaridade, conforme explicado na Figura 4.22. O gráfico com a distribuição das características exclusivas de cada família é apresentado na Figura 4.25.

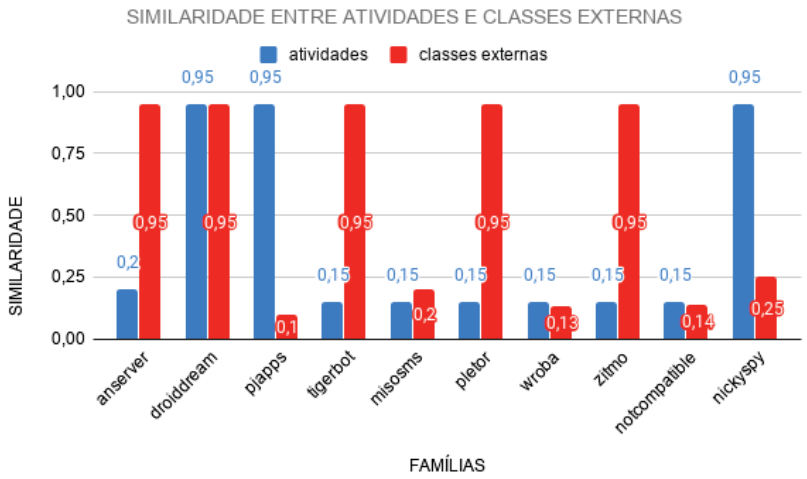


Figura 4.23: Similaridade entre os conjuntos de serviços, receptores e permissões das famílias

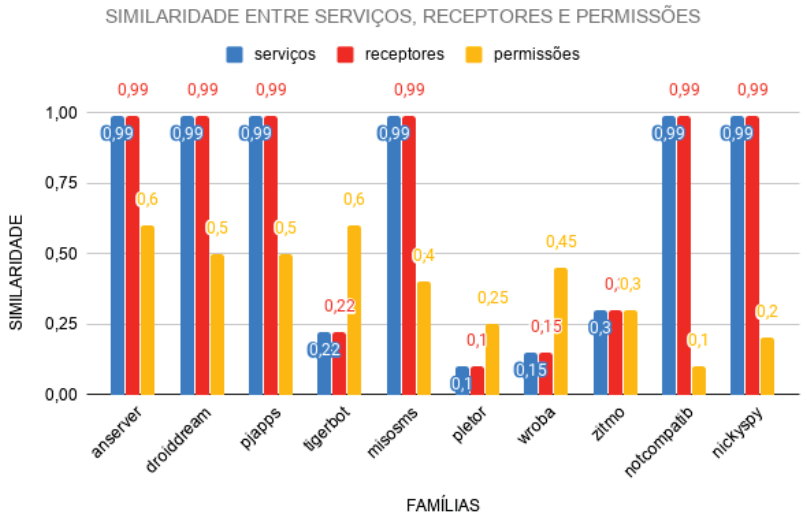


Figura 4.24: Similaridade entre os conjuntos de serviços, receptores e permissões das famílias

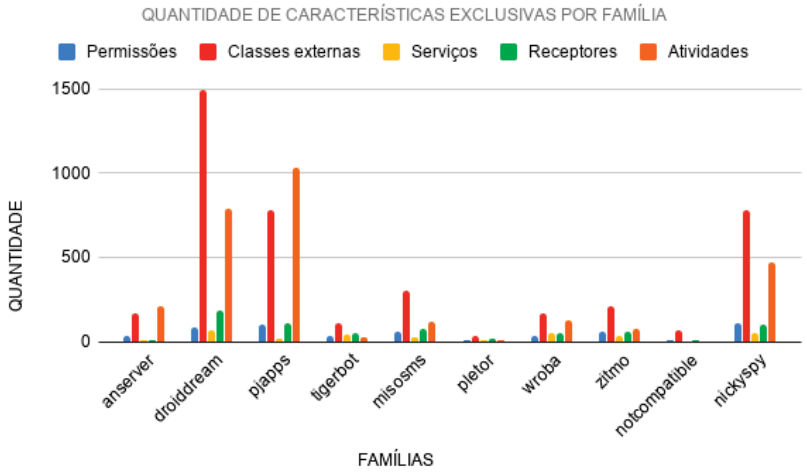


Figura 4.25: Total de características exclusivas por Família

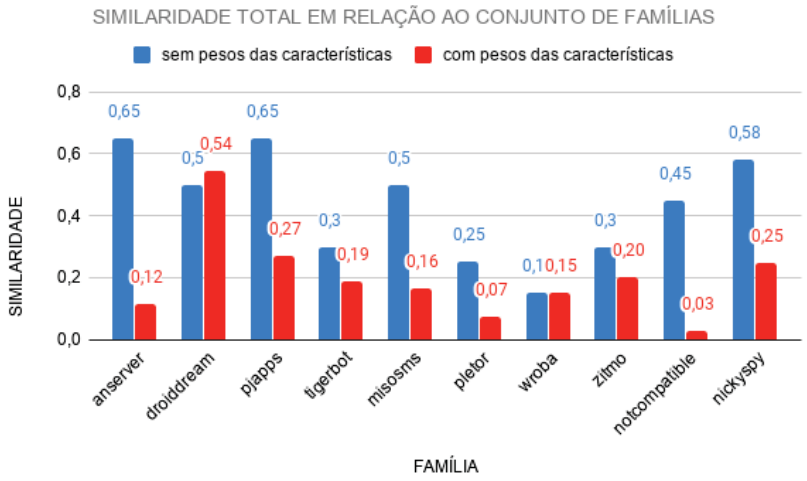


Figura 4.26: Similaridade total em relação ao conjunto de famílias

Constatou-se que as maiores quantidades de caraterísticas exclusivas foram encontradas nas famílias Droiddream e PJApps, enquanto as menores quantidades foram das famílias Pletor e a NotCompatible (Figura 4.25). Além disso, é importante ressaltar que os exemplares da família NotComplatible demonstraram técnicas de ofuscação, dificultando a extração de características.

Já a Figura 4.26 apresenta o gráfico da similaridade existente entre as famílias, utilizando os pesos das características (colunas direitas) e aplicando apenas o cálculo de similaridade (colunas esquerdas).

As colunas esquerdas indicam os valores de similaridades entre as famílias antes do uso de pesos em conjunto com o cálculo da similaridade (índice de Jaccard). Observando os valores sem pesos das características, foi possível verificar no gráfico (Figura 4.26), que diversas famílias apresentavam grande valor de similaridade entre elas, como a Anserver e a PJapps, Droiddream e MisoSMS, a Zitmo e a Tigerbot, e assim por diante.

Entretanto, após o uso do peso no cálculo da similaridade, ainda houve confusão entre famílias, como: MisoSMS e Wroba, Tigerbot e Zitmo, PJapps e Nickyspy (colunas direitas da Figura 4.26).

Tabela 4.16: Comparação da qualidade dos agrupamentos finais, por método de ligação e quantidade de amostras, através de índices de validação

	Single Linkage				Average Linkage				Complete Linkage			
Amostras	F-Score	Precisão	Recall	Pureza	F-Score	Precisão	Recall	Pureza	F-Score	Precisão	Recall	Pureza
20	**0,99**	**0,99**	**0,99**	**0,99**	0,97	0,97	0,97	0,97	0,87	0,87	0,87	0,87
50	**0,98**	**0,98**	**0,98**	**0,98**	0,95	0,95	0,95	0,95	0,85	0,85	0,85	0,85
70	**0,97**	**0,97**	**0,97**	**0,97**	0,94	0,94	0,94	0,94	0,84	0,84	0,84	0,84

Após análise dos resultados, verificou-se que os maiores valores dos índices de qualidade foram obtidos através do método de Ligação Simples, com precisão, *recall* e *f-score* de, aproximadamente, 99%. Já os menores valores foram obtidos nos experimentos com 70 amostras, utilizando o método de Ligação Completa (aproximadamente 84 % de precisão).

4.6 CONSIDERAÇÕES FINAIS SOBRE OS EXPERIMENTOS

Em relação ao conjunto de amostras não-maliciosas, notou-se que algumas amostras de aplicativos semelhantes estão em grupos diferentes. Por exemplo, *por que os aplicativos de comunicação ficaram em grupos separados?* As amostras desses aplicativos têm número heterogêneo de atributos, e algumas permissões são encontradas apenas em alguns aplicativos (por exemplo, o aplicativo Google Meet possui vários receptores de entrada e saída de reunião e de compartilhamento de dados). Na análise de atributos do Whats app, foram encontrados vários recursos de backup, gerenciamento de listas de bloqueio de contatos, funcionalidade de autenticação em duas etapas e ações de pagamento. Outra constatação foi que nem todos os aplicativos têm permissão para usar mensagens SMS, como observado no Google Meet e no Telegram.

malware* vs *goodware Examinando as características das versões dos aplicativos não maliciosos investigados, verificou-se que podem ser menos confusos para determinar a linhagem (ancestralidade) das amostras, devido ao grande número de recursos comuns entre as amostras.

Em relação à rotulação de conjuntos de dados, construir bases para aplicativos benignos é menos complexo do que para códigos maliciosos. O uso de antivírus para verificar se um arquivo é malicioso ou não é diferente do processo de rotulagem de amostra com classificação em famílias. Além disso, na análise de *malware* ainda há a verificação se uma variável é código novo ou se foi apenas modificado e reempacotado.

Atributos Estáticos A classificação de *malware* apenas por meio de análise estática tem limitações, como já mencionado em (Moser et al., 2007). Ainda assim, as permissões são ferramentas úteis para detectar comportamentos perigosos e indesejados (por exemplo, todos os aplicativos móveis maliciosos analisados tiveram as seguintes permissões): `KILL_BACKGROUND_PROCESSES`, `REAL_GET_TASKS`, `BLUETOOTH_ADMIN`, e `READ_LOGS`.

Durante a análise dos atributos, ficou evidente que aplicativos maliciosos e não maliciosos utilizam permissões potencialmente prejudiciais à segurança do usuário. No entanto, como você sabe o que o aplicativo pretende quando solicita permissão para gravar áudio? Esse consentimento possibilita o compartilhamento de mensagens de áudio entre pessoas não autorizadas ou espionar as conversas da vítima e usar os áudios para extorsão? Outras permissões consideradas críticas apareceram em diversos goodware:

- `WRITE_EXTERNAL_STORAGE` : Youtube, Whats app, Telegram, Messenger, Instagram, Garena Free Fire, Netflix, e Tik Tok.

- ACCESS_FINE_LOCATION : Youtube, Whats app, Telegram, Messenger e Instagram.
- ACCESS_COARSE_LOCATION: Youtube, Whats app, Telegram e Messenger.
- GET_TASKS:Whats app, Tik Tok e Garena Free Fire.
- CHANGE_WIFI_STATE: Whats app, Messenger e Garena Free Fire.
- READ_PHONE_STATE: Youtube, Whats app, Telegram, Messenger, Instagram, Garena Free Fire e Netflix.

Algumas permissões apareciam apenas em código malicioso, como DISABLE_KEYGUARD (FaketaoBao, Android.spy.277, FakeApp_AL, Android Defender e AvForAndroid) e READ_HISTORY_BOOKMARKS (FakeApp e AndroidDefender).

Em relação a outros atributos, verificou-se que as famílias Android Defender e FakeApp_AL possuem serviços de monitoramento de bateria. As amostras fakeAV e faketaoBao apresentaram receptores de mudança de status de Wifi. As amostras FakeJobOffer têm atributos relacionados ao áudio, enquanto o AvForAndroid monitora as atividades dos leitores de livros e pdfs. Além disso, as famílias android.spy.277 e FakeApp_AL estão interessadas nas ações do usuário ao manipular imagens e vídeos. Os exemplares de Virus Shield e AvPass apresentaram atividades e serviços de pagamento.

Como limitação, a medida de similaridade e os recursos escolhidos foram insuficientes para distinguir as famílias de *malware* e alguns casos de goodware-*malware*, como jogos, também.

4.7 DEFINIÇÃO DE UM FLUXO DE ATIVIDADES PARA AGRUPAMENTO DE *MALWARE*

Diante da ampla diversidade de conceitos e técnicas existentes na classificação de variantes de *malware*, filogenia e análise de agrupamentos, é comum surgirem dúvidas que, geralmente, não são esclarecidas pela literatura. Entre as incertezas encontram-se opções de algoritmos de agrupamento, escolha de medidas de similaridade, impacto da natureza de variáveis, necessidade e formas de seleção de características, entre outras.

Depois de analisar diversos trabalhos correlatos sobre o assunto, verificou-se que apesar de serem encontradas inúmeras publicações utilizando agrupamento de famílias de *malware*, existe uma lacuna de orientações das melhores práticas. Essas orientações podem incluir a seleção de atributos, formas de representação de dados, escolhas de medidas de similaridade e de algoritmos de agrupamento, e os impactos dessas escolhas sobre os resultados dos agrupamentos.

Desse modo, foi desenvolvido um fluxo de atividades de aprendizado de máquina para agrupamento com quatro fases diferentes, o qual é apresentado na Figura 4.27

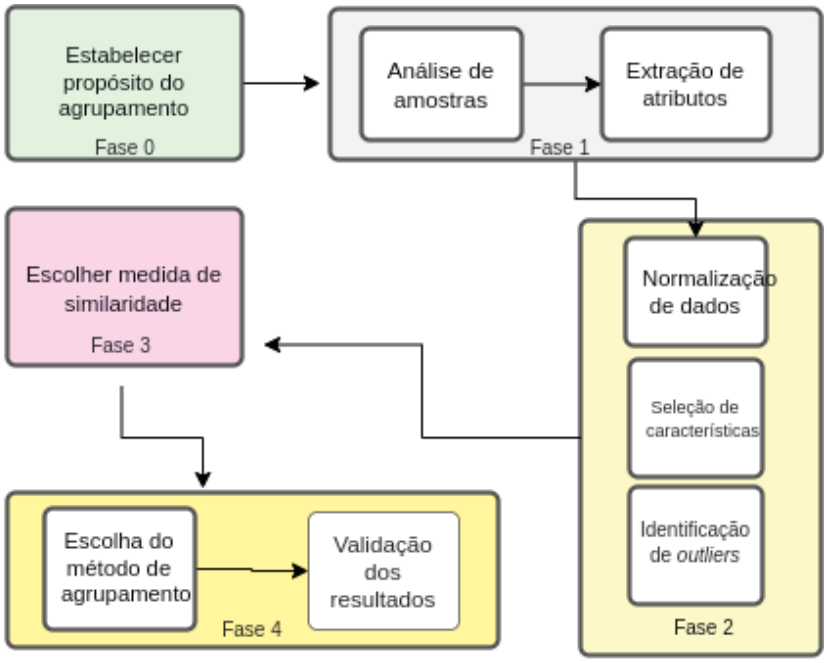


Figura 4.27: Método sugerido

- **Fase 0 - Estabelecer o propósito da classificação** A análise e agrupamento de amostras de *malware* são realizados com diferentes finalidades. Por exemplo, os fabricantes de dispositivos de hardware podem estar interessados em investigar grandes quantidades de amostras de códigos maliciosos para identificar possíveis ameaças a produtos físicos. Um analista de segurança de rede pode analisar *malware*, como ransomware Android, para responder a incidentes. Outros propósitos incluem o desenvolvimento de soluções *antimalware*, pesquisa de linhagem de software malicioso e a caça a *malware* de "dia zero". Diferentes combinações de tipo de análise, algoritmos de agrupamento e de medidas de similaridade dependem da finalidade do agrupamento, do tamanho do conjunto de dados, do poder computacional e do tempo disponível.

- **Fase 1 - Análise de amostras e extração de características**

  De acordo com a extensão da coleção de exemplares de código malicioso, como base de arquivos *.apks*, o poder computacional e o tempo disponível, o analista experimenta e define o tipo de análise (Capítulo 2.7).

  Após definir e realizar a análise das amostras, são extraídos os atributos e definidos os tipos de representação dos dados.

- **Fase 2 - Normalização de dados e outras tarefas**

  Normalização e padronização de Dados: A natureza dos atributos (discreto, binário, contínuo) e a escala variável (nominal, ordinal) afetam a estrutura dos *clusters*. Assim, tarefas como padronização ou normalização de dados podem beneficiar a qualidade dos resultados. A padronização tende a minimizar problemas de dispersão, normalizar recursos de acordo com um critério específico e eliminar ruídos e redundâncias para melhorar a precisão dos resultados. Alguns métodos disponíveis incluem Z- Score, Amplitude Máxima e Min-Max.

  Seleção de atributos: Anteriormente destacou-se que a seleção de variáveis pode melhorar a qualidade dos resultados usando métodos supervisionados e não supervi-

sionados. Também foi mencionado (Capítulo 2.7.5) que nem todos os métodos de seleção de recursos se aplicam ao agrupamento. A escolha do tipo de seleção, usando filtros, *wrappers* ou outra abordagem, também depende do objetivo da análise, do poder computacional e do tempo disponível para esta etapa de pré-agrupamento. Os filtros não dependem dos algoritmos de agrupamento, apesar dos *wrappers* geralmente fornecerem resultados mais satisfatórios. Uma das alternativas recomendadas é combinar as duas técnicas quando possível. Nos experimentos de análise de *malware* de (Babaagba e Adesanya, 2019), a acurácia aumentou de 54,3 para 74,4% através de seleções de características antes do uso do algoritmo Expectativa-Maximização.

Identificação de *Outliers*: *Outliers* são objetos considerados diferentes e inconsistentes com o resto do conjunto. Essas anomalias podem afetar negativamente o resultado ou ser exatamente o que está sendo buscado (Dutta et al., 2012). Na análise de *malware*, os *outliers* podem trazer informações importantes, como identificar amostras rotuladas incorretamente. Geralmente, no agrupamento assume-se que a maioria dos elementos está em agrupamentos mais densos. Desse modo, são investigados como *outliers* dados que não pertencem a nenhum grupo ou pertencem a pequenos *clusters*. As estratégias para lidar com anomalias incluem exclusão de elementos inconsistentes e análise separada do conjunto anômalo.

- **Fase 3 - Escolha da medida de similaridade** Determinar a melhor medida de similaridade não é uma tarefa trivial, dependendo diretamente dos dados analisados. Vários estudos e pesquisas sobre medidas de distância e de similaridade estão disponíveis na literatura, como nas pesquisas de Irani et al. (2016), Bisandu et al. (2019) e Pfitzner et al. (2009). Assim, fica a pergunta: como decidir? A opção padrão geralmente é a distância euclidiana, embora essa medida não seja ideal em diversos casos. Geralmente, quando o analista compreende os dados do conjunto de amostras, ele testa várias medidas e verifica visualmente quais apresentaram maior consistência com o padrão dos elementos do conjunto de dados. Por exemplo, um teste que pode ser realizado é: operando com textos na forma de *strings*, com a similaridade Cosseno, Índice de Jaccard ou Métrica de Levenshtein, utilizando amostras da mesma família, qual métrica indicou similaridade mais perto da real entre as amostras já rotuladas? E o que acontece com amostras de famílias dissimilares usando essas mesmas medidas?

  Embora existam dezenas/centenas de equações e de medidas de similaridade, a Tabela 4.17 apresenta recomendações baseadas na literatura analisada e no tipo de dados.

Tabela 4.17: Métricas e tipos de atributos/características

Tipo	**Medidas e índices**
Strings	Hamming, métrica Levenshtein, NCD, Cosseno
Documentos	distância Dice, Cosseno, índice de Jaccard
Vetor binário	índice de Jaccard, distância Hamming, distância euclidiana
Grafos	Máximo subgrafo comum, métrica Levenshtein
Imagens	distância euclidiana, Manhattan, VCAD, distância de histogramas
Modelos ocultos de Markov	divergência Kullback-Leibler

- **Fase 4 - Seleção do método de agrupamento e validação de resultados**

Conforme mencionado na Seção 2.8, cada tipo de agrupamento tem vantagens e desvantagens.

Por exemplo, métodos hierárquicos são vantajosos quando se sabe que os dados têm relacionamentos hierárquicos. Porém, esses métodos não lidam bem com *outliers*, não sendo adequados para conjuntos extensos de dados. Além disso, geralmente é difícil definir o número ideal de *clusters* analisando visualmente o dendrograma. Por exemplo, o método de Expectativa Máxima não é indicado para encontrar *clusters* significativamente pequenos, como quando o conjunto de dados possui famílias com apenas uma ou poucas amostras.

Métodos de partições, como K-means, são rápidos e altamente escaláveis. No entanto, eles dependem do número inicial de *clusters* e são adequados para conjuntos de dados com formas esféricas. À medida que o número de características aumenta, a escalabilidade do algoritmo diminui. Também é sensível a *outliers*, por isso indica-se a normalização dos dados.

Algoritmos baseados em densidade são ótimos para dados com alta dimensionalidade e com diferentes formas de conjuntos. Além disso, eles são uma ótima opção para conjuntos de dados ruidosos, porque esse tipo lida bem com valores discrepantes. No entanto, eles geralmente sofrem de *overfitting*, isto é, esses algoritmos podem memorizar o que fazer a partir dos dados de treinamento, não apresentando bons resultados quando aplicados a outros conjunto de dados para testes.

Na Tabela 4.18 são apresentados exemplos de propósitos de agrupamento e de escolhas associadas.

Tabela 4.18: Exemplos de propósitos de agrupamento de *malware*.

Propósito	**Número de amostras**	**Tipo de Análise**	**Métrica de Similaridade/Distância**	**Agrupament**
Investigar famílias com o mesmo ancestral	milhões	Híbrida - chamadas a API Java	Índice de Jaccard	UPGMA
Detectar *malware*	milhares	Dinâmica - CFG	distância Edit	K-means e K-
Análise de relatórios sobre famílias de *malware*	milhares	Estática - TFIDF	NCD	DBSCAN
Verificar permissões perigosas frequentes	milhares	Estática - Permissões	distância euclidiana	Self-Organizi
Identificar comportamentos potenciais de *malware* bancários	centenas	Dinâmica - dados de rede	distância euclidiana normalizada	Fuzzy C-Mea

5 ANDROIDGYNY - VISÃO GERAL DO *FRAMEWORK* PROPOSTO PARA CLASSIFICAÇÃO DE VARIANTES DE *MALWARE* ANDROID EM FAMÍLIAS

Nos capítulos anteriores foram apresentadas etapas e resultados de atividades que incluíram:

- Revisão de mais de 70 trabalhos correlatos e de suas configurações de experimentos
- Avaliação de cenários envolvendo análise de medidas de similaridade/dissimilaridade, algoritmos de agrupamento
- Comparação do uso de amostras não-maliciosas (*goodware*) em conjunto com *malware*
- Uso de características exclusivas de famílias conhecidas, cálculos de pesos desses atributos exclusivos no cálculo da similaridade entre os elementos das famílias e comparação com medóides
- Definição de um fluxo básico de atividades de agrupamento de *malware*

Partindo disso, foi desenvolvido o Androidgyny, um *framework* para análise de amostras, extração de características, detecção de maliciosidade em *apks* e classificação de variantes de *malware* focados na plataforma Android.

A seguir é apresentada a sua estrutura, seus componentes e funcionamento.

5.1 VISÃO GERAL DO ANDROIDGYNY

A Figura 5.1 apresenta a visão geral do *framework* desenvolvido.

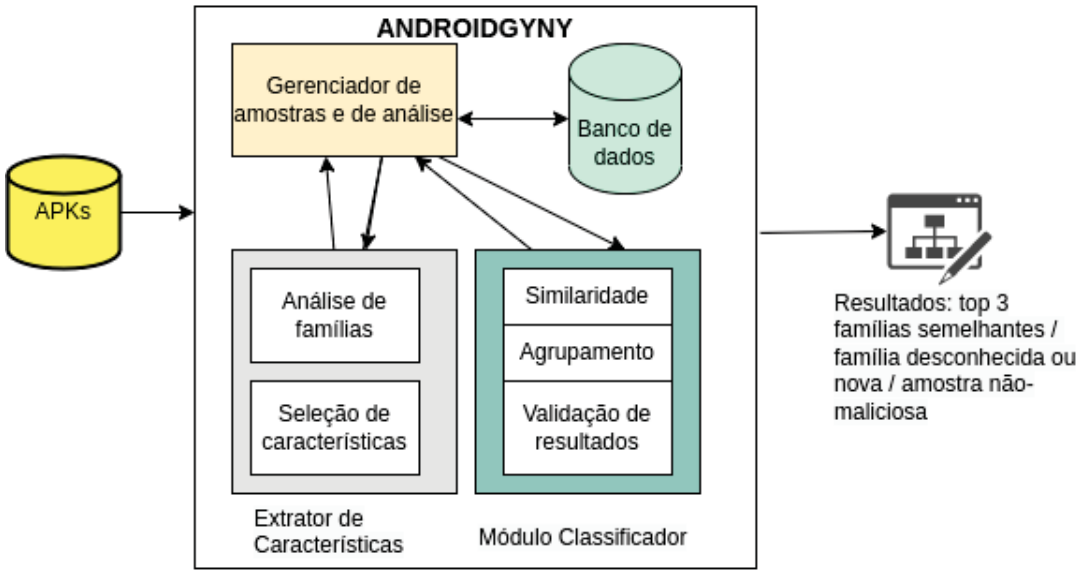


Figura 5.1: Componentes do protótipo Androidgyny

São quatro componentes principais do Androidgyny: gerenciador de análise, o extrator de características, o módulo classificador e o banco de dados:

- **Gerenciador de amostras e de análise:** O componente Gerenciador de amostras e de análise é responsável por controlar cada análise, receber as amostras maliciosas ou não-maliciosas, controlar fluxo de dados entre todos os componentes e realizar a conexão com o banco de dados.

- **Extrator de características:** O componente Extrator de características analisa cada *apk*, coleta os recursos necessários e envia as informações de volta para o Gerenciador de amostras e de análise armazenar no banco de dados.

- **Módulo Classificador:** O Módulo Classificador recupera características relevantes do banco de dados para cada aplicativo, verificando a similaridade com as famílias conhecidas por meio do algoritmo de agrupamento. Finalmente, envia as resultantes para o Gerenciador de amostras e de análise para armazenamento no banco de dados.

- **Banco de dados:** O banco de dados é usado para armazenar e recuperar informações do conjunto de amostras e dos resultados da classificação. O resultado de cada análise é apresentado ao final da análise.

Informações mais detalhadas sobre o funcionamento de cada componente são apresentadas a seguir.

5.1.1 Gerenciador de amostras e de análise

Esse componente é o módulo principal do protótipo desenvolvido, sendo o ponto inicial para cada experimento. Podem ser realizadas as seguintes atividades: extração de características, classificação da amostra, ou ambas.

Caso haja erros no módulo Extrator de Características, devido a amostras corrompidas ou com mecanismos de criptografia avançados, esses resultados também são inseridos na base de dados. Além de amostras maliciosas, amostras não-maliciosas (*goodware*) também podem ser analisadas.

5.1.2 Extrator de Características

Para extração de características foi utilizada a API do Apktool, por meio de descompactação do arquivo *.apk* para coletar informações estáticas das amostras inseridas.

A quantidade de características extraídas de cada *apk*, de cada variante e de cada família não são heterogêneas. Todavia, o conjunto total de registros de cada experimento é extenso. Por exemplo, na base de dados atual do Androidgyny foram registradas descrições únicas de 617 nomes de famílias, 16.290 permissões, 39.235 receptores, 58.243 serviços, 224.190 classes externas e 857.074 atividades. Desse modo, com restrições de poder computacional e de tempo, para otimizar o desempenho da classificação, foram selecionadas as características mais importantes para o conjunto de famílias de cada experimento.

Os atributos utilizados foram o conjunto de permissões, atividades, serviços, receptores e de classes externadas chamadas pelo aplicativo. Após a extração de características, realizada via análise estática dos arquivos, realizam-se as seguintes verificações:

- **Atualização de registros de características:** para as características investigadas, são consultadas se as descrições já existem na base de dados. Caso contrário, são inseridas nas tabelas correspondentes.

- **Seleção de atributos:** Devido à quantidade extensa de atributos dos exemplares, para otimizar a classificação e o vetor binário que seria utilizado, foram elencadas as características mais importantes, através de verificação dos atributos utilizados por todas as famílias do *dataset* de cada experimento. Esse processo de seleção primeiramente foi realizado via consultas ao banco de dados.

5.2 MÓDULO CLASSIFICADOR

Após receber informações do componente Gerenciador de amostras e de análise, o Módulo Classificador realiza verificações de maliciosidade, de análise de similaridade com famílias já conhecidas (base de dados) e se há características/comportamentos desconhecidos.

A análise de similaridade é realizada através do método de agrupamento: o algoritmo hierárquico com K-medoids foi adaptado e é apresentado no Algoritmo 1.

Algoritmo 1: Agrupamento hierárquico com medóides

Entrada: X é um conjunto de dados com n pontos, k é o número desejado de agrupamentos

Saída: M = $\{m_1,...,m_k\}$ é um conjunto de k medóides dos agrupamentos; conjunto de agrupamentos resultantes G

```
1  início
2      M = Calcular os medóides iniciais(X,k) ;
3      repita
4          para i = 1,...,n faça
5              Calcular a distância da instância X_i em relação aos elementos do
                conjunto de medóides iniciais M ;
6              Encontrar os pares com as menores distâncias (agrupamento
                hierárquico) ;
7              Mover a instância para o grupo G_k do medóide mais próximo ;
8              M = Recalcular os medóides para cada grupo G;
9          fim
10     até o número de agrupamentos ser constante;
11 fim
```

Na Figura 5.2 podem ser visualizados alguns medóides de famílias já cadastradas, com o valor médio em comparação aos outros elementos da família.

```
tsrpimenta@tsrpimenta-Lenovo-IdeaPad-S145-15API:~/Documents/modeloF$ python3 print_medoide.py
select fam_desc from familias where id = 104
valor médio de similaridade: 0.24193548387096775
select distinct f.nome from medoide_fam f where f.id_fam = 104 and f.contador = 0.24193548387096775
Familia: PUA.AndroidOS.Ewind
Nome do apk: 7c6f09574962328ddedb1b2e962eec7c


select fam_desc from familias where id = 577
valor médio de similaridade: 0.02501441753171857
select distinct f.nome from medoide_fam f where f.id_fam = 577 and f.contador = 0.02501441753171857
Familia: feiwo
Nome do apk: a8fe9d0365b5779a43282848565d074a


select fam_desc from familias where id = 607
valor médio de similaridade: 0.2808219178082192
select distinct f.nome from medoide_fam f where f.id_fam = 607 and f.contador = 0.2808219178082192
Familia: Biige
Nome do apk: 7cb70612e7ca5d14dc8b6a6ac835c341


select fam_desc from familias where id = 608
valor médio de similaridade: 0.15178571428571427
select distinct f.nome from medoide_fam f where f.id_fam = 608 and f.contador = 0.15178571428571427
Familia: FakeInst
Nome do apk: 4f9a18692f7adbfcaedb0e6f4fdecc9a
```

Figura 5.2: Exemplo de medóides de famílias

Geração dos *clusters* iniciais: Inicialmente, com n famílias conhecidas, são criados n grupos iniciais, cada *cluster* com o elemento que melhor representa a família. Com a inserção de novas variantes, os *clusters* são atualizados e os medóides são recalculados. O algoritmo K-medoids adaptado pode ser visualizado no Algoritmo 1. Após a realização de diversos testes com K-means, k-medoids e agrupamento hierárquico de ligação média, decidiu-se pelo uso do K-medoids, pela robustez para lidar com os medóides. Diferentemente das escolhas aleatórias de pontos como medóides, no presente método o elemento é selecionado de acordo com a quantidade de características que possui em relação aos demais da família. Utiliza-se o índice de Jaccard para comparar os conjuntos de características dos elementos de cada família, sendo escolhida a amostra que possui a maior similaridade média em relação aos demais.

5.2.1 Módulo Classificador e a Avaliação de Variantes

A Figura apresenta o fluxograma dos tipos de resultados possíveis da classificação.

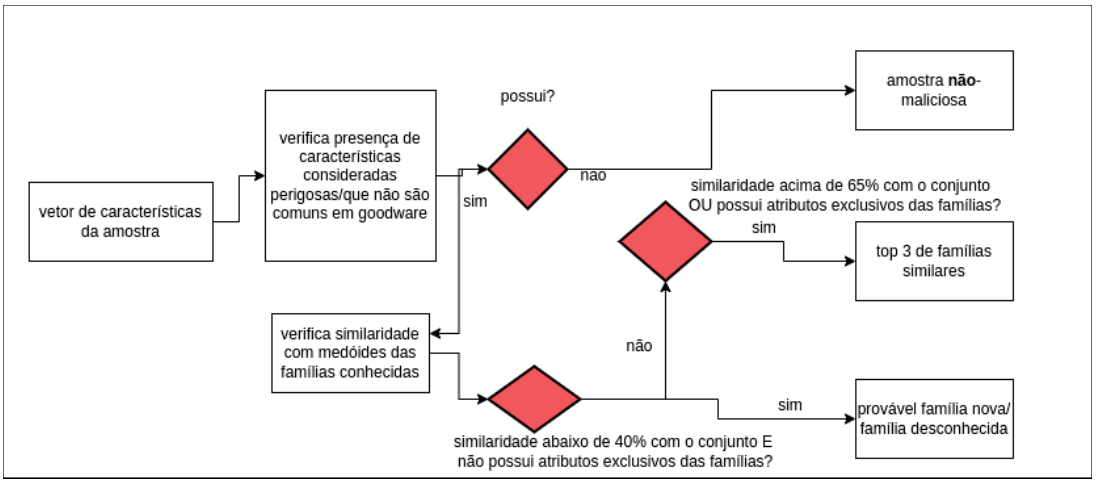


Figura 5.3: Algoritmo básico da decisão do resultado

A partir dos resultados da comparação de similaridade com os medóides, encontram-se as seguintes situações:

- **Amostra não-maliciosa:** De acordo com as verificações de características de alto e médio risco de maliciosidade, presentes nas famílias conhecidas e ausentes nas amostras não-maliciosas, o Módulo Classificador indica se houve maliciosidade encontrada na amostra.

- **Variantes desconhecidas de famílias conhecidas**: são verificadas se as amostras analisadas possuem características exclusivas de alguma família já estudada ou grande similaridade com algum representante (medóide). Caso positivo, a amostra é considerada uma variante da família já rotulada.

- **Família desconhecida**: Se não for encontrada nenhuma característica exclusiva, realiza-se agrupamento do elemento com os medóides das famílias já armazenadas, indicando as três famílias (top 3) com maior similaridade. Caso os valores de similaridade com todos os medóides conhecidos forem baixos, como por exemplo, abaixo de 40%, a amostra é colocada em um *cluster* novo. Para esse grupo de exemplares desconhecidos, são utilizados outros algoritmos de agrupamento com os vetores de características de todas as amostras da base de dados. Dependendo do tamanho da base de características, o processo pode ser demorado. Para essa etapa foram realizados testes com os algoritmos K-means e com agrupamento aglomerativa hierárquico (HAC), conforme apresentado nos experimentos anteriores. Como o HAC apresenta a desvantagem de tempo, devido ao cálculo da matriz de similaridades ou de distâncias, optou-se pelo K-means.

- **Avaliação dos resultados:** Como as amostras são rotuladas em arquivos separados *dataset*, são calculadas as taxas de rotulação corretas e incorretas para cada família analisada. Além disso, as seguintes métricas foram utilizadas para cálculos da qualidade dos agrupamentos resultantes:

  - **Pureza(W,C,D)** = $\frac{1}{N} * \Sigma_k max_j(n_k \cap c_j)$, onde D é o conjunto de amostras de treinamento, N é a soma das amostras em D, W= N1, N2..., Nm denota o conjunto de *clusters* e C=c1,c2...,cj é o conjunto de classes, oriundas dos rótulos de *ground-truth* dos dados de treinamento, nk é o *cluster* k e cj é a classe j.
  - **Precisão** = $\frac{TP}{TP+FP}$
  - **Recall** = $\frac{TP}{TP+FN}$
  - **F1** = $\frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$

  Após o retorno das informações do Módulo Classificador é realizada a validação dos resultados. Para essa etapa, são feitas buscas do nome do arquivo analisado e dos *hashes* MD5, SHA-1 e SHA256 no Virus Total.

6 ANDROIDGYNY - EXPERIMENTOS

Neste capítulo são apresentados os experimentos realizados para avaliação do Androidgyny, incluindo informações sobre as configurações de cada experimento, famílias, quantidade de amostras, características e métricas utilizadas.

6.1 CONFIGURAÇÃO DOS EXPERIMENTOS

Os experimentos foram executados em uma máquina com as seguintes configurações:

- **Processador:** AMD Ryzen 5 35000U
- **Memória RAM:** 12 GB
- **Armazenamento local:** 1 TB
- **Armazenamento extra:** 3 HDs portáteis Seagate Expansion com 3 TB de capacidade (cada)
- **Sistema Operacional:** Ubuntu 20.04.2 LTS

Androidgyny foi implementado utilizando Python v3.6, Flask v1.02 e jinj2 v2.10. Foi utilizado servidor de banco de dados MySQL para armazenamento dos resultados. Outros módulos requeridos foram Androguard v3.2.1, Scikit-learn v0.20, Tensorflow v1.13.1, e Keras v2.2.4.

6.2 EXPERIMENTO 1 - COMPARAÇÃO COM O GEFDROID

Durante a pesquisa de trabalhos correlatos, verificou-se que a replicação da maioria dos estudos analisados não é um processo simples. Entre a literatura estudada, o trabalho de Fan et al. (2019b) foi um dos raros trabalhos em que os códigos-fonte estavam disponíveis publicamente. Além disso, os autores também disponibilizaram amostras maliciosas de três famílias que foram utilizadas para classificar *malware* Android em famílias. Os autores utilizaram chamadas a APIs em subgrafos e algoritmos de detecção de comunidade para agrupar exemplares não rotulados.

Tabela 6.1: Informações sobre as famílias e grupos do *dataset* CICAndMal2017

Grupo	**Famílias**	**Total**
Adware	Dowgin,Ewind,Feiwo,Gooligan,Kemoge,Koodous, Mobidash,Selfmite,Shuanet,Youmi	104
Scareware	AndroidDefender,AndroidSpy.277,AV for Android,AVpass,FakeApp, FakeApp.AL,FakeAV,FakeJobOffer, FakeTaoBao,Penetho,VirusShield	112
Ransomware	Charger,Jisut,Koler,LockerPin,Simplocker, Pletor,PornDroid,RansomBO,Svpeng,WannaLocker	101
PremiumSMS	Beanbot,Fakemart,Jifake,Mazarbot,Zsone	48
SMS	Biige,FakeInst,FakeNotify,Nandrobox,Plankton,SMSsniffer	61

Na Tabela 6.2 estão disponibilizados os resultados de Pureza, Recall, Precisão e F-score do método proposto e do GEFDROID.

Tabela 6.2: Experimentos com GEFDROID e proposta atual

Grupo	**Pureza**		**Precisão**		**Recall**		**F-score**	
	Gefdroid	**Nosso**	**Gefdroid**	**Nosso**	**Gefdroid**	**Nosso**	**Gefdroid**	**Nosso**
Adware	0,7	1	0,74	0,96	0,61	1	0,67	0,97
Scareware	0,57	0,96	0,60	0,91	0,49	1	0,55	0,94
Ransomware	0,85	1	0,89	1	0,74	1	0,82	1
PremiumSMS	0,87	1	0,92	1	0,76	1	0,84	1
SMS	0,8	1	0,85	1	0,7	1	0,77	1

Os índices para o conjunto de *malware* dos tipos Ransomware e PremiumSMS foram os que apresentaram os melhores resultados para ambos os métodos. A maior diferença entre os resultados foi para o tipo Scareware, onde a pureza do GEFDROID ficou em 0,57 e do nosso método, 0,96. Possivelmente isso se deve devido ao alto número de características exclusivas que foram encontradas em famílias desse grupo.

6.3 EXPERIMENTO 2

Nesse experimento foram selecionadas 25 das famílias com maior quantidade de exemplares, do repositório Androzoo. O conjunto de amostras é composto por 1.000 exemplares, escolhidos aleatoriamente, totalizando 25 mil exemplares. Na Tabela 6.3 estão disponibilizados os nomes das famílias e suas quantidades de atributos exclusivos.

Tabela 6.3: Quantidade de características exclusivas de cada família

Família	**Permissões**	**Atividades**	**Classes**	**Serviços**	**Receptores**
Admogo	9	1335	939	0	0
Adwo	28	5328	3810	0	0
Airpush	190	5226	3063	22	26
Anydown	43	2779	1367	0	1
Artemis	188	15599	7968	42	36
Deng	151	10334	6071	0	0
Domob	18	5025	3578	16	22
Dowgin	125	10884	10632	45	49
Droidkungfu	8	2917	2107	9	6
Feiwo	11	2484	2022	46	25
Genpua	142	5625	3629	0	0
Geyser	990	2361	5161	21	12
Ginmaster	67	8526	5836	0	0
Inmobi	611	6124	8843	26	48
Jiagu	361	36186	679	0	0
Kuguo	26	4605	6366	38	34
Leadbolt	72	3425	1977	0	0
Plankton	83	4216	1842	11	3
Revmob	484	1367	1139	0	2
Skymobi	67	3613	5315	7	11
Smspay	138	0	9457	0	0
Umeng	30	781	6665	0	0
Waps	44	2889	5933	0	0
Wooboo	3	2549	1711	0	0
Youmi	56	6294	461	39	36

6.4 RESULTADOS DAS CARACTERÍSTICAS ANALISADAS

O conjunto total de características armazenados para os exemplares das 25 famílias foi de 202.475 atividades, 133.696 classes externas, 4.726 permissões, 396 serviços e 390 receptores.

Destaca-se que serviços e receptores não são utilizados por parte das famílias analisadas. Além disso, verifica-se que os números de atividades e classes externas são expressivos, sendo acima de 1000 chamadas à classes em mais de 20 famílias.

6.4.1 Permissões

As permissões mais utilizadas pelas 25 famílias são apresentadas na Tabela 6.4.

Tabela 6.4: Top 20 permissões mais utilizadas pelas famílias maliciosas analisadas

Nº	**Descrição**
1	android.permission.INTERNET
2	android.permission.ACCESS_NETWORK_STATE
3	android.permission.READ_PHONE_STATE
4	android.permission.WRITE_EXTERNAL_STORAGE
5	android.permission.WAKE_LOCK
6	android.permission.ACCESS_WIFI_STATE
7	android.permission.VIBRATE
8	android.permission.GET_TASKS
9	android.permission.RECEIVE_BOOT_COMPLETED
10	android.permission.SYSTEM_ALERT_WINDOW
11	com.android.launcher.permission.INSTALL_SHORTCUT
12	android.permission.MOUNT_UNMOUNT_FILESYSTEMS
13	android.permission.CHANGE_WIFI_STATE
14	android.permission.READ_EXTERNAL_STORAGE
15	android.permission.READ_LOGS
16	android.permission.CAMERA
17	android.permission.RECORD_AUDIO
18	android.permission.WRITE_SETTINGS
19	android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
20	android.permission.RESTART_PACKAGES

Nota-se que a grande maioria das permissões são popularmente encontradas em aplicações maliciosas ou com problemas de privacidade de dados. No entanto, as permissões menos frequentes são as dos itens 19 e 20.

As maiores taxas de semelhança entre permissões de famílias foram encontradas entre as famílias Umeng, Wooboo, Artemis e Deng, com exemplares apresentando entre 86 e 88% de similaridade entre os conjuntos desses atributos.

6.4.2 Serviços

Na Tabela 6.5 encontra-se a relação dos 20 serviços mais visualizados nas famílias analisadas.

Tabela 6.5: Top 20 serviços mais utilizados pelas famílias maliciosas analisadas

Nº	Descrição
1	com.inmobi.commons.internal.ActivityRecognitionManager
2	com.ironsource.mobilcore.MobileCoreReport
3	com.qbiki.seattleclouds.ExpansionFilesDownloaderService
4	com.qbiki.geofencing.ReceiveTransitionsIntentService
5	com.qbiki.modules.videolist.DownloadService
6	com.kuguo.ad.MainService
7	com.apperhand.device.android.AndroidSDKProvider
8	com.airpush.android.PushService
9	net.youmi.android.AdService
10	com.umeng.common.net.DownloadingService
11	com.senddroid.AdService
12	com.qihoo.util.CommonService
13	com.ringcrop.record.RecorderService
14	com.skymobi.pay.common.service.PayCtrlService
15	com.ringcrop.player.service.MediaPlayerService
16	com.ringcrop.service.DownMCService
17	com.taobao.accs.ChannelService
18	com.taobao.accs.data.MsgDistributeService
19	com.taobao.accs.ChannelService$KernelService
20	org.android.agoo.accs.AgooService

Em relação aos serviços, apenas exemplares das famílias Inmobi,Artemis,Domob e Airpush utilizavam esse tipo de recurso. Entre os itens mais utilizados, nomes de outras famílias aparecem na descrição de diversos serviços, como Kuguo, Youmi, Umeng, Senddroid, Skymobi, entre outras.

6.4.3 Receptores

A Tabela 6.6 apresenta os 20 principais receptores utilizados pelos exemplares.

Tabela 6.6: Top 20 receptores mais utilizados pelas famílias maliciosas analisadas

Nº	Descrição
1	com.google.android.gcm.GCMBroadcastReceiver
2	com.inmobi.commons.core.utilities.uid.ImIdShareBroadCastReceiver
3	com.qbiki.gcm.GCMBroadcastReceiver
4	com.qbiki.seattleclouds.ExpansionFilesDownloaderAlarmReceiver
5	com.ironsource.mobilcore.InstallationTracker
6	com.kuguo.ad.MainReceiver
7	com.qbiki.modules.goaltracker.GoalTrackerAlarmReceiver
8	com.airpush.android.MessageReceiver
9	com.airpush.android.DeliveryReceiver
10	com.airpush.android.UserDetailsReceiver
11	com.google.android.c2dm.C2DMBroadcastReceiver
12	net.youmi.android.AdReceiver
13	com.qihoo.util.CommonReceiver
14	com.fw.ttze.receiver.FwBReceiver
15	com.taobao.accs.EventReceiver
16	com.igexin.sdk.PushReceiver
17	com.ringcrop.aNewCrop.receiver.NetworkReceiver
18	com.ringcrop.receiver.PhoneStatusReceiver
19	com.ringcrop.aNewCrop.receiver.AutoStartReceiver
20	com.ringcrop.aNewCrop.receiver.PhoneStateReceiver

Ao analisar os nomes dos receptores utilizados, verifica-se que alguns deles contém nomes de famílias de *malware*, como Inmobi (item 2), Kuguo (item 6), Airpush (itens 8,9 e 10), Igexin (item 16), entre outros. Além disso, das 25 famílias analisadas neste experimento, apenas 14 utilizavam receptores. As que não utilizavam são listadas a seguir: Admogo, Adwo, Deng, Genpua, Ginmaster, Jiagu, Leadbolt, Smspay, Umeng, Waps e Wooboo.

6.4.4 Atividades

Diferentemente dos serviços, recursos do tipo Atividades são amplamente utilizados por estas famílias. A Tabela 6.7 apresenta as 20 principais.

Tabela 6.7: Top 20 atividades mais utilizadas pelas famílias maliciosas analisadas

Nº	Descrição
1	com.google.ads.AdActivity
2	com.google.android.gms.ads.AdActivity
3	cn.domob.android.ads.DomobActivity
4	com.vpon.adon.android.WebInApp
5	com.adwo.adsdk.AdwoAdBrowserActivity
6	net.youmi.android.AdActivity
7	com.startapp.android.publish.list3d.List3DActivity
8	com.tencent.tauth.AuthActivity
9	com.startapp.android.publish.AppWallActivity
10	com.revmob.ads.fullscreen.FullscreenActivity
11	com.baidu.mobads.AppActivity
12	com.adsmogo.adview.AdsMogoWebView
13	com.chartboost.sdk.CBImpressionActivity
14	com.facebook.ads.InterstitialAdActivity
15	com.jirbo.adcolony.AdColonyFullscreen
16	com.jirbo.adcolony.AdColonyOverlay
17	com.jirbo.adcolony.AdColonyBrowser
18	com.tencent.connect.common.AssistActivity
19	com.google.android.gms.ads.purchase.InAppPurchaseActivity
20	com.millennialmedia.android.VideoPlayer

No conjunto de atividades nota-se intuitos diversificados, incluindo itens de anúncios, compras, eventos de tela e *players* de vídeos.

Ressalta-se que as atividades utilizadas por este *dataset*, além da grande quantidade e diversidade, apresentaram pouca similaridade entre as famílias. O valor máximo de semelhança obtido desse tipo de característica foi de 19,29%, entre as famílias Adwo e Admogo.

6.4.5 Classes externas

A Tabela 6.8 apresenta as 20 principais classes externas mais chamadas pelos exemplares.

Tabela 6.8: Top 20 classes externas mais utilizadas pelas famílias maliciosas analisadas

Nº	Descrição
1	`Landroid/support/v4/util/AtomicFile`
2	`Landroid/support/v4/content/FileProvider`
3	`Landroid/support/v4/content/FileProvider$SimplePathStrategy`
4	`Landroid/support/v4/content/ContextCompat`
5	`Landroid/support/v4/os/EnvironmentCompat;`
6	`Lcom/google/ads/util/g`
7	`Lcom/google/ads/util/g$b`
8	`Lcn/domob/android/a/f`
9	`Lcn/domob/android/a/b`
10	`Landroid/support/v4/provider/RawDocumentFile`
11	`Lcn/domob/android/a/g`
12	`Lcom/ironsource/mobilcore/j`
13	`Lorg/apache/http/entity/mime/content/FileBody`
14	`Lcom/ironsource/mobilcore/l`
15	`Lcom/google/android/gms/internal/zzac`
16	`Lcom/google/android/gms/internal/zzv`
17	`Lcom/google/android/gms/tagmanager/ak`
18	`Lcom/google/android/gms/tagmanager/v$a`
19	`Lcom/google/android/gms/internal/zzal`
20	`Lcom/ironsource/mobilcore/i`

A quantidade de chamadas a classes externas frequentemente possuem maior utilização em aplicativos maliciosos em comparação aos não maliciosos. Por exemplo, nas famílias Artemis e Smspay, exemplares chegavam a usar dezenas de milhares de classes.

6.4.6 Resultados das análises de similaridade das famílias

Em relação à validação dos resultados do agrupamento, na Tabela 6.9 são apresentadas as taxas de erros e acertos para cada família.

Tabela 6.9: Taxas de erros e acertos para cada família, onde CR significa corretamente rotulados e IR, incorretamente rotulados.

Família	C.R	I.R	Confusão
Admogo	984	16	Waps e Domob
Adwo	663	337	Wooboo, Droidkungfu, Waps e Ginmaster
Airpush	988	12	Revmob, Plankton e Wooboo
Anydown	989	11	Genpua e Waps
Artemis	865	135	Jiagu e Deng
Deng	840	160	Waps
Domob	961	39	Waps e Wooboo
Dowgin	842	158	Waps e Ginmaster
Droidkungfu	810	190	Ginmaster, Wooboo, Waps e Kuguo
Feiwo	998	2	Kuguo
Genpua	998	2	Waps
Geyser	997	3	Waps e Leadbolt
Ginmaster	994	6	Waps
Inmobi	968	32	Waps e Ginmaster

Jiagu	995	5	Waps e Kuguo
Kuguo	994	6	Waps
Leadbolt	992	8	Waps
Plankton	998	2	Umeng
Revmob	945	55	Waps
Skymobi	996	4	Waps e Kuguo
Smspay	991	9	Waps, Genpua e Ginmaster
Umeng	997	3	Wooboo
Waps	947	53	Ginmaster
Wooboo	984	16	Admogo,Waps, Domob e Leadbolt
Youmi	926	74	Wooboo,Waps,Domob e Ginmaster

As taxas de acertos mais altas foram dos exemplares das famílias Feiwo, Genpua e Plankton, apresentando cerca de 98% de acerto. A maior taxa de amostras incorretamente agrupadas foi da família Adwo, com aproximadamente 33% de erros. Além disso, percebe-se que entre as famílias que mais aparecem como rótulo incorreto estão Waps e Wooboo.

Como mencionado anteriormente na descrição do método de agrupamento, quando feita a comparação com os medóides, o resultado apresenta as três famílias mais similares à amostra analisada. Desse modo, na Tabela 6.10 são disponibilizadas informações sobre as famílias e o número de permissões similares. Essas informações foram processadas a partir das informações do agrupamento das amostras com os medóides.

Tabela 6.10: Permissões e Famílias Similares

Família	Top 1		Top 2		Top 3	
	Família	Nº	Família	Nº	Família	Nº
Admogo	Deng	84	Waps	82	Artemis	82
Adwo	Jiagu	152	Artemis	151	Smspay	150
Airpush	Umeng	121	Artemis	120	Jiagu	118
Anydown	Deng	82	Genpua	79	Jiagu	79
Artemis	Jiagu	199	Smspay	181	Genpua	162
Deng	Artemis	157	Smspay	151	Jiagu	148
Domob	Artemis	132	Dowgin	114	Umeng	113
Dowgin	Jiagu	157	Artemis	148	Smspay	146
Droidkungfu	Skymobi	93	Artemis	93	Waps	92
Feiwo	Jiagu	177	Dowgin	131	Artemis	130
Genpua	Jiagu	168	Smspay	168	Artemis	162
Geyser	Skymobi	16	Inmobi	15	Airpush	15
Ginmaster	Jiagu	124	Dowgin	123	Artemis	122
Inmobi	Artemis	118	Dowgin	117	Jiagu	115
Jiagu	Artemis	199	Smspay	189	Feiwo	177
Kuguo	Deng	103	Smspay	102	Dowgin	101
Leadbolt	Deng	118	Umeng	116	Artemis	115
Plankton	Artemis	112	Umeng	105	Leadbolt	104
Revmob	Deng	71	Artemis	70	Leadbolt	70
Skymobi	Smspay	155	Artemis	145	Jiagu	135
Smspay	Artemis	181	Genpua	166	Umeng	147
Umeng	Jiagu	155	Artemis	153	Smspay	147
Waps	Artemis	127	Deng	124	Smspay	122
Wooboo	Umeng	92	Artemis	91	Deng	90
Youmi	Jiagu	161	Smspay	158	Artemis	154

A pequena quantidade de permissões similares da Geyser, mesmo em comparação com as três famílias mais semelhantes à ela, confirma a grande dissimilaridade com as demais famílias. Por exemplo, foi verificada similaridade abaixo de 1%, no conjunto de características, com as amostras da Inmobi, Skymobi e Airpush.

Em relação ao conjunto de permissões utilizadas, 88,46% permissões da família Umeng eram iguais às da família Wooboo, sendo a taxa mais alta encontrada neste experimento.

Na Tabela 6.11 são disponibilizadas famílias semelhantes e o número de atividades similares.

Tabela 6.11: Atividades e Famílias Similares

Família	Top 1		Top 2		Top 3	
	Família	Nº	Família	Nº	Família	Nº
Admogo	Adwo	411	Domob	230	Waps	223
Adwo	Droidkungfu	774	Waps	535	Artemis	427
Airpush	Plankton	323	Inmobi	285	Youmi	284
Anydown	Genpua	302	Inmobi	138	Artemis	129
Artemis	Smspay	1053	Deng	1007	Genpua	934
Deng	Artemis	1007	Smspay	982	Skymobi	557
Domob	Dowgin	441	Droidkungfu	393	Adwo	357
Dowgin	Jiagu	701	Artemis	599	Kuguo	466
Droidkungfu	Adwo	774	Skymobi	680	Domob	393
Feiwo	Dowgin	406	Jiagu	371	Artemis	330
Genpua	Artemis	934	Smspay	461	Dowgin	357
Geyser	Inmobi	17	Airpush	15	Genpua	15
Ginmaster	Dowgin	424	Artemis	403	Deng	373
Inmobi	Airpush	285	Jiagu	263	Genpua	253
Jiagu	Artemis	832	Dowgin	701	Smspay	511
Kuguo	Dowgin	466	Deng	458	Artemis	417
Leadbolt	Airpush	274	Plankton	208	Inmobi	203
Plankton	Airpush	323	Dowgin	291	Youmi	230
Revmob	Airpush	238	Inmobi	180	Artemis	164
Skymobi	Smspay	1195	Artemis	680	Jiagu	557
Smspay	Skymobi	1195	Artemis	1053	Deng	982
Umeng	Dowgin	386	Artemis	359	Waps	322
Waps	Adwo	535	Artemis	505	Youmi	423
Wooboo	Adwo	384	Droidkungfu	326	Waps	287
Youmi	Waps	423	Adwo	410	Artemis	345

Em relação às atividades, os rótulos que mais são mencionados como semelhantes são Artemis, Dowgin, Adwo, Inmobi, Smspay e Waps. Anydown, Feiwo, Leadbolt e Umeng, que foram mencionadas no ranking de semelhança das permissões, não são citadas nessa relação.

Na Tabela 6.12 podem ser visualizadas relacionamentos entre famílias semelhantes e o número de classes externas em comum.

Tabela 6.12: Classes Externas e Famílias Similares

Família		Top 1		Top 2		Top 3
	Família	Nº	Família	Nº	Família	Nº
Admogo	Adwo	1870	Waps	1363	Domob	1244
Adwo	Waps	2790	Droidkungfu	2409	Smspay	2162
Airpush	Youmi	1373	Leadbolt	1339	Revmob	1337
Anydown	Genpua	1418	Inmobi	1022	Artemis	1004
Artemis	Smspay	17058	Deng	3068	Dowgin	2944
Deng	Artemis	3068	Smspay	2770	Dowgin	2221
Domob	Adwo	2007	Droidkungfu	1764	Waps	1613
Dowgin	Smspay	3031	Artemis	2944	Ginmaster	2300
Droidkungfu	Adwo	2409	Waps	1840	Youmi	1448
Feiwo	Dowgin	1411	Smspay	1384	Artemis	1260
Genpua	Artemis	2295	Dowgin	1787	Smspay	1715
Geyser	Inmobi	136	Artemis	134	Genpua	131
Ginmaster	Dowgin	2300	Artemis	2191	Smspay	2106
Inmobi	Artemis	1773	Genpua	1524	Dowgin	1444
Jiagu	Smspay	102	Umeng	78	Artemis	71
Kuguo	Dowgin	1939	Waps	1937	Adwo	1026
Leadbolt	Airpush	1339	Inmobi	1113	Artemis	1026
Plankton	Dowgin	929	Airpush	860	Leadbolt	788
Revmob	Airpush	1337	Inmobi	1111	Leadbolt	985
Skymobi	Smspay	2776	Dowgin	1914	Artemis	1891
Smspay	Artemis	3230	Dowgin	3031	Skymobi	2776
Umeng	Smspay	2270	Dowgin	2252	Artemis	2085
Waps	Adwo	2790	Youmi	2060	Kuguo	1937
Wooboo	Adwo	1160	Droidkungfu	1061	Waps	984
Youmi	Adwo	2103	Waps	2060	Artemis	1580

Analisando os números relacionados às classes externas em comum verifica-se que Artemis, Dowgin, Adwo, Smspay e Waps continuam como os rótulos mais mencionados. A comparação entre o número de menções das famílias pode ser visualizada na Figura 6.1.

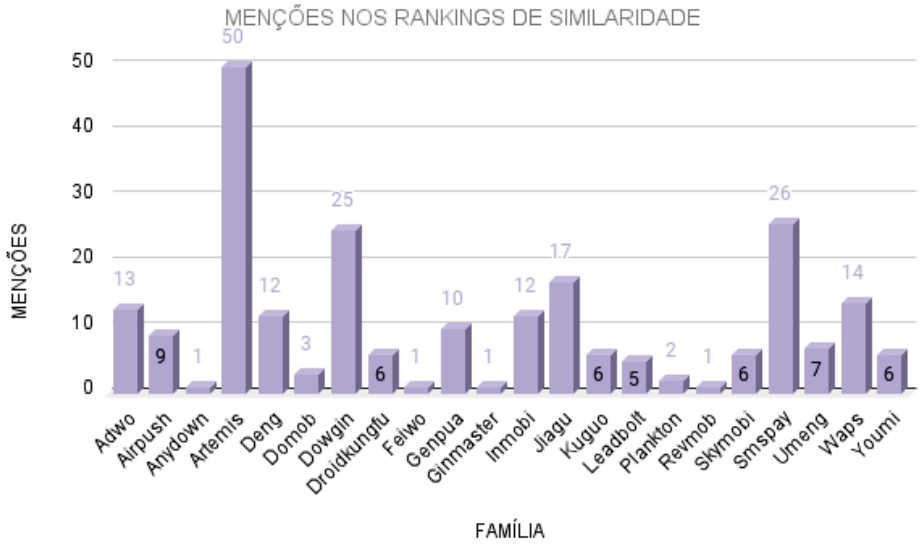


Figura 6.1: Menções das famílias nos *rankings* de similaridades

Em relação ao total de menções de similaridade, a família Artemis foi a que mais foi referenciada, com 50 menções, seguida pelas famílias Smspay e Dowgin. Algumas famílias não foram referenciadas como semelhantes, por isso não apareceram no gráfico da Figura 6.1.

6.4.7 Considerações sobre análise de similaridade das famílias

De acordo com as informações da base de dados, são sumarizadas os seguintes dados sobre as famílias:

Admogo: A taxa de acerto no agrupamento das amostras da Admogo chegou a 98%. Os exemplares incorretamente rotulados foram agrupados com variantes das famílias Domob e Waps. A confusão pode ser explicada pela alta similaridade com as duas famílias, como visto na análise de Atividades (Waps), Classes (Domob)e Permissões (ambas).

Adwo: A maior taxa de erro neste experimento foi de amostras da família Adwo, aproximadamente 33% de agrupamento incorreto. Essa família grupo foi um dos que mais apresentou similaridade com grande número de outras famílias, como Droidkungfu, Waps, Smspay, Artemis e Jiagu. Algumas variantes também foram classificadas como da família Wooboo. Apesar de não aparecer nos Top 3 de similaridade, 84,6% das permissões de Wooboo estão no conjunto de permissões da Adwo, o mesmo sendo verificado para 41,5% das classes externas.

Airpush: Apenas 12 amostras rotuladas como da família Airpush foram agrupadas erroneamente neste experimento, em *clusters* de exemplares das famílias Plankton, Wooboo e Revmob. Duas dessas famílias aparecem nos top 3 de similaridade utilizando Atividades (Plankton) e Classes externas (Revmob). Além disso, 78,8% das permissões e 16% das classes externas de Wooboo são equivalentes ao conjunto da Airpush.

Anydown: A taxa de acerto da Anydown foi de mais de 98%. Genpua aparece como semelhante nos três rankings: atividades, classes externas e permissões, o que pode ter influenciado na confusão do agrupamento. Os valores de similaridade de Anydown e Waps não foram tão altos no contexto de atividades e classes externas, porém 51,7% das permissões dessa família estão contempladas no conjunto da família Waps.

Artemis: Uma das famílias mais citadas como semelhante, possivelmente sendo ancestral comum a outras do conjunto. Dos 1000 exemplares, 135 foram agrupados com variantes de Jiagu e Deng. A confusão entre essas famílias é justificada pela grande similaridade em permissões, atividades e classes externas.

Deng:A taxa de acerto no agrupamento foi de 84%, tendo 160 exemplares agrupados como variantes da família Waps. Apesar dessa família não aparecer nas três mais parecidas com a Deng, elas têm 124 permissões em comum.

Domob: Assim como alguns exemplares foram classificados erroneamente como Wooboo, algumas amostras dessa família foram agrupadas como Domob. Dowgin aparece como uma das famílias mais parecidas, com permissões e atividades. A confusão com variantes da família Waps pode estar relacionada ao conjunto de classes externas, apresentando 1613 em comum com a Domob.

Dowgin: Quase 16% dos 1000 exemplares de Dowgin foram agrupados com variantes de Ginmaster e Waps. A proximidade com a Ginmaster aparece nos três rankings de similaridade, destacando a quantidade de 2300 classes em comum entre essas duas famílias. Além disso, Jiagu e Artemis se destacam como famílias próximas a Dowgin.

Droidkungfu: As famílias Waps e Skymobi se destacam na similaridade com essa família. A taxa de acerto no agrupamento das variantes foi de 81%, porém os exemplares foram espalhados em grupos de variantes de Ginmaster, Wooboo, Waps e Kuguo.

Feiwo: Uma das maiores taxas de acerto foi do agrupamento de exemplares de Feiwo, tendo apenas duas das 1000 amostras sendo classificadas como Kuguo. Apesar de não ser clara a semelhança entre as duas famílias, 79% das permissões de Feiwo estão no conjunto de atributos da Kuguo.

Genpua: Apesar de não haver confusão com a família Smspay, a semelhança entre as duas famílias é significativa. Apenas duas amostras foram agrupadas como variantes de Waps, sendo assim uma das menores taxas de erro deste experimento.

Geyser: Como mencionado anteriormente, a família Geyser é uma das mais dissimilares ao dataset, apresentando taxas baixas de semelhança com famílias como Genpua, Inmobi e Airpush. O fato de ter atributos tão distintos contribuiu para a taxa de erro de apenas 3%.

Ginmaster: Apesar das famílias Dowgin e Artemis possuírem um número significativo de atributos equivalentes a Ginmaster, os seis exemplares agrupados erroneamente foram classificados como Waps. Porém, também houve confusão de amostras de Waps agrupadas como variantes de Ginmaster, confirmando a similaridade entre as duas famílias.

Inmobi: As maiores semelhanças foram encontradas com exemplares das famílias Artemis, Dowgin e Jiagu. Entretanto, houve confusão no agrupamento com variantes de Waps e Ginmaster, totalizando taxa de erro de aproximadamente 3%.

Jiagu: A acurácia na classificação das amostras foi de 99%, entretanto cinco amostras da família Jiagu foram categorizadas como Waps e Kuguo. Apesar disso, as maiores similaridades foram verificadas com Smspay e Artemis.

Kuguo: Do total de 1000 exemplares, seis foram agrupados com amostras de Waps. Uma explicação é a existência de 1937 classes externas em comum entre Waps e Kuguo. Outra família com semelhança significativa é a Dowgin.

Leadbolt: Houve confusão de oito exemplares de Leadbolt classificados como Waps. Apesar de não estar nas três mais parecidas, 42% das permissões de Leadbolt estão no conjunto da Waps também. Artemis e Inmobi aparecem duas vezes nos rankings de semelhanças com a Leadbolt.

Plankton: Com uma das menores taxas de erro, tendo apenas 2 amostras foram rotuladas como Umeng. Uma possibilidade da confusão é a existência de 105 permissões em comum entre as duas famílias. Dowgin, Airpush e Leadbolt aparecem mais de uma vez no ranking de similaridade com essa família.

Revmob: Cerca de 18% das classes externas da família Revmob estão no conjunto da família Waps, possivelmente explicando os 55 exemplares de confusão com a última. Além disso, Leadbolt e Inmobi apresentam semelhança significativa com essa família.

Skymobi: A taxa de acerto no agrupamento de variantes da Skymobi foi de mais de 99%, tendo confusão com Waps e Kuguo. A similaridade com Kuguo foi de 33,9 % das permissões em comum com Kuguo e 10% das classes externas. Já com a família Waps, foram verificadas semelhanças entre 40,9% das permissões e 14,6% das classes externas. Todavia, Artemis, Smspay e Jiagu aparecem mais de uma vez nos rankings de semelhança com essa família.

Smspay: Skymobi e Artemis se sobressaem nos rankings das famílias similares à Smspay. Entretanto, houve confusão de nove exemplares sendo categorizados como Waps, Genpua e Ginmaster. Apesar de não estarem nas três mais parecidas, destaca-se que há 12% de classes em comum com Ginmaster e 23% em comum com a família Genpua. A similaridade com Waps chega a 63% no conjunto de permissões.

Umeng: Dowgin, Smspay e Artemis apresentam quantidades significativas de atributos em comum com a família Umeng. Em relação aos acertos e erros no agrupamento, apenas três exemplares foram confundidos como variantes de Wooboo. No ranking de similaridade constata-se que Wooboo tem 92 permissões em comum com a família Umeng.

Waps: Mais de 50 exemplares foram categorizados como Ginmaster. Uma das possíveis causas da confusão 15% das classes externas de 53% das permissões encontradas em amostras da Waps foram vistas nos conjuntos de atributos da Ginmaster.

Wooboo: Houve confusão de 16 amostras de Wooboo com variantes de Admogo, Waps, Domob e Leadbolt. Verifica-se que das permissões de Wooboo, 83% aparecem em Waps, 79% em Domob, 76,9% em Leadbolt e 64,4% no conjunto da família Admogo. Além disso, Waps aparece duas vezes nos rankings de semelhança de Wooboo (classes externas e atividades).

Youmi: Com taxa de acerto de mais de 92%, no agrupamento de variantes de Youmi, 74 foram clusterizados com Wooboo, Waps, Domob e Ginmaster. Entre as famílias mais similares, Waps tem 2060 de classes externas e 423 atividades em comum com Youmi. Em relação ao conjunto de classes externas, há semelhança em 20,3% com Domob, 11,5% com Wooboo e 14,5% com Ginmaster. Já no conjunto de

permissões, 42,9% são encontradas também nos exemplares de Domob, 43,7% em Ginmaster e 35,7% em Wooboo.

7 DISCUSSÃO

As seguintes questões foram estabelecidas para nortear esta pesquisa:

QP1- Como as métricas influem na separação das famílias?

Conforme analisado nos experimentos realizados, em especial no Experimento 2 (Cap. 4), através da análise visual das árvores geradas foi possível verificar diferenças entre os *clusters* resultantes do uso de métricas distintas. Uma das influências mais significativas percebidas foi no total de *clusters* criados em cada árvore, a partir do mesmo algoritmo de agrupamento hierárquico, mesmos atributos e tipo de ligação utilizados. Desse modo, também verificou-se que a natureza dos dados e a forma de representação de dados estão diretamente relacionados à métrica utilizada e a qualidade dos resultados obtidos.

Além da influência mencionada, como existem complexidades na rotulação de famílias e variantes de *malware*, atributos em comum, alto grau de compartilhamento de estruturas e funcionalidades, a separação das amostras em famílias é impactada, resultando em variantes da mesma família sendo agrupadas em *clusters* separados.

QP2- Por que o agrupamento falha para determinados rótulos/famílias?

Como já discutido anteriormente, a classificação de softwares não maliciosos, incluindo sistemas operacionais, aplicações bancárias, comunicadores instantâneos, geralmente é menos complexa em comparação ao agrupamento de códigos maliciosos. Uma possível explicação é o desenvolvimento de softwares comerciais por empresas diferentes, utilizando linguagens de programação distintas, grande quantidade de chamadas a APIs e também de bibliotecas próprias. Para exemplificar, como demonstrado no Experimento 3, exemplares do Google Meet e do Whats App foram corretamente separados. Apesar de estarem na mesma categoria de aplicativos e de usarem os mesmos tipos de recursos dos dispositivos, incluindo microfone, gravação de áudio e compartilhamento de localização em tempo real, elementos como atividades, serviços, nomes de classes, quantidade de linhas de código e ícones utilizados podem diferenciar os binários dos softwares. Um exemplo da alta dissimilaridade em aplicativos não-maliciosos com funcionalidade similares foi apresentado no Experimento 3 (Capítulo 4): quando comparados os conjuntos de características de amostras do Instagram e do Tik Tok, foram detectadas diferenças em 355 atividades, 1238 classes, 66 serviços, 32 receptores e 53 permissões.

Infelizmente, para o cenário de classificação de *malware* em famílias, há categorias que compartilham estruturas e funcionalidades utilizadas por outros grupos. Por exemplo, no segundo experimento de validação do *framework* Androidgyny, verificou-se que 84,6% das permissões da família Wooboo estão no conjunto de permissões da Adwo, o mesmo sendo verificado para 41,5% das classes externas utilizadas. Ainda em relação à Wooboo, 78,8% das permissões e 16% das classes externas de Wooboo são equivalentes ao conjunto da Airpush. Desse modo, devido ao extenso reuso de atributos, módulos, funções e código-fonte, técnicas de agrupamento podem falhar para determinadas famílias.

Resultados de outras análises realizadas são apresentados a seguir:

- **Análise dos dendrogramas gerados (árvores)** Apesar de existirem técnicas para corte automático (poda) dos ramos das árvores resultados dos agrupamentos hierárquicos, a maioria das análises são realizadas visualmente por analistas humanos. Isso demanda esforços como estudo e tempo. Todavia, durante a análise visual dos dendrogramas do Experimento 1, disponíveis no Apêndice, sobressaiu-se a importância do uso do Coeficiente de Correlação Cofenético para análise dos tipos de ligações e medidas utilizadas. Quanto mais próximo de 1, melhor o método de agrupamento, sendo que valores abaixo de 0,8 indicam distorções significativas na árvore obtida.

- **Análise das medidas de similaridade:** Como resultados do Experimento 1 verificou-se que a combinação do índice de Jaccard com o método de ligação simples apresentou os coeficientes de correlação cofenética menos satisfatórios para o conjunto de dados analisados, com valores

menores que 0,8. Entre os maiores resultados dos coeficientes prevaleceram o método de ligação média. Já em relação à medida de similaridade, Cosseno (*Cosine*) e Jaro Winkler podem ser considerados os mais apropriados para comparação com a representação de dados escolhida (n-grams e *strings*).

Outros resultados significativos foram os usos de medidas de similaridade, como a distância de Levenshtein e índice de Jaccard adaptado com pesos dos atributos exclusivos. A primeira teve desempenho suficiente para uso com *strings*, enquanto a segunda apresentou desempenho superior após o uso de pesos, conforme explicado no Experimento 3.

- **Análise dos algoritmos de agrupamento:** No terceiro experimento foram utilizados medóides, elementos representativos das famílias analisadas como agrupamentos iniciais. Como vantagem desse método verificou-se que o cálculo da similaridade foi mais rápido, em comparação ao cálculo da matriz de distâncias com todos os elementos do conjunto. Como ponto negativo, como etapa inicial deve ser encontrado o elemento que melhor representa o conjunto, demandando análise prévia das características da família.

- **Análise das famílias:** A análise de famílias antes do agrupamento pode ser útil em casos nos quais o analista tem apenas parte dos rótulos do conjunto de amostras; outra indicação é como etapa anterior à classificação a partir de métodos supervisionados. Entretanto, com a falta de tempo e de *datasets* atualizados, rótulos inconsistentes e novas famílias desconhecidas aos softwares antimalware, esse tipo de análise nem sempre é possível de ser realizado.

- **Uso de *Goodware*:** Examinando as características das versões dos aplicativos não maliciosos investigados, verificou-se que podem ser menos confusos para determinar a linhagem (ancestralidade) das amostras, devido ao grande número de recursos comuns entre as amostras.

  Em relação à rotulação de conjuntos de dados, construir bases para aplicativos benignos é menos complexo do que para códigos maliciosos. O uso de antivírus para verificar se um arquivo é malicioso ou não é diferente do processo de rotulagem de amostra com classificação em famílias. Além disso, na análise de *malware* ainda há a verificação se uma variável é código novo ou se foi apenas modificado e reempacotado.

- **Uso de diferentes métricas de validação**: Conforme mencionado no estudo de trabalhos correlatos, existem diversas medidas de avaliação da qualidade dos agrupamentos gerados. Por exemplo, no primeiro experimento o Coeficiente de Correlação Cofenético foi útil para avaliar qual método de ligação gerou o dendrograma mais apropriado ao conjunto de exemplares agrupado. Já no Experimento 3, os valores das métricas de Acurácia, Precisão e outras medidas de qualidade foram compatíveis com os resultados visualizados nos gráficos do algoritmo K-means, por exemplo.

- **Análise das características utilizadas:** Nos quatro experimentos realizados foram utilizadas características estáticas, as quais são comentadas na Tabela 7.1.

Tabela 7.1: Análise das características utilizadas

Característica	Observações	Exp.
Opcodes	Opcodes são úteis para diferenciar códigos maliciosos de não maliciosos. Para distinção de famílias de *malware* é necessário realizar mais experimentos em relação a acurácia com a forma de representação utilizada. Por exemplo, diferenças de desempenho utilizando *2-gram, 3-gram* e assim por diante.	1 e 2
Chamadas a APIs	Concluiu-se que o uso de chamadas a APIs e opcodes foi superior ao uso de permissões para distinção de famílias.	1
Permissões	O uso apenas das permissões foi o menos satisfatório para o agrupamento das famílias de *malware*, enquanto o uso de chamadas a APIs e de opcodes teve resultados equivalentes	Todos
Atividades	Os atributos de atividades utilizadas pelas aplicações maliciosas, apesar de grande quantidade presente nos exemplares maliciosos, a similaridade entre os conjuntos de famílias diferentes era baixa, não sendo possível realizar distinção adequada entre elas apenas com esse atributo.	3 e 4 e experimentos do Androidgyny
Serviços	Como diversas famílias não utilizavam serviços, esses atributos não devem ser considerados como principais critérios para distinção de famílias ou de aplicações maliciosas/não-maliciosas.	3 e 4 e experimentos do Androidgyny
Receptores	Como diversas famílias não utilizavam receptores, assim como no caso dos serviços, esses atributos não devem ser considerados como principais critérios para distinção de famílias ou de aplicações maliciosas/não-maliciosas	3 e 4 e experimentos do Androidgyny
Classes externas	Nas aplicações maliciosas, geralmente, a quantidade de classes chamadas era numerosas, podendo ser superior a mil. Todavia, as chamadas não eram suficientemente distintas para separar adequadamente famílias. Em relação à classificação *goodware-malware*, grande número de nomes das classes maliciosas apresentavam sequências de letras e caracteres incomuns, como vogais ou consoantes repetidas em sequência, como por exemplo na classe `LYyyyyyyyyyyyvyvv`	3 e 4 e experimentos do Androidgyny

O uso de características concatenadas, como Opcodes + Permissões, não se mostrou mais eficaz que apenas o uso de um tipo de característica. Além disso, a concatenação de características diferentes apresentou resultados ligeiramente inferiores na maioria dos casos.

7.1 ANÁLISE DA VIABILIDADE DO USO DE ALGORITMOS DE ESCALONAMENTO NA FILOGENIA DE *MALWARE*

Nos últimos anos, com a pandemia da COVID-19, a filogenia tem recebido destaque nas pesquisas e na mídia. Todavia, o estudo da linhagem e da ancestralidade das espécies têm sido realizados há décadas. A determinação do ancestral comum, dos descendentes, comportamentos, formas de replicação ou de reprodução e outras informações das espécies são essenciais para realizar previsões de mutações, desenvolver vacinas e entender padrões relevantes.

Diversas técnicas têm sido aplicadas no estudo de linhagens, como nos casos das variantes do Coronavírus, quanto para objetos, animais e códigos de computador. No entanto, dependendo dos

elementos a serem classificados e da amplitude do critério escolhido para o agrupamento, os resultados podem ser insatisfatórios. Por exemplo, na separação de seres vivos utilizando um critério genérico, como "ter asas", podem estar no mesmo grupo pássaros, insetos, morcegos e até dinossauros. Todavia, a pesquisa e determinação do melhor critério de separação para o cenário pode ser demasiadamente demorada, inviabilizando todo o processo de categorização.

No contexto de software, o estudo da linhagem e da similaridade de códigos é valioso para áreas como Computação Forense e investigação de pirataria. Já no estudo de *malware*, da mesma forma como acontece na biologia, parte do material de um código malicioso é passado para suas novas variantes. Assim como a seleção natural dos seres vivos mais aptos, os códigos mais aprimorados sobrevivem, neste caso, não sendo detectados pelas defesas antimalware.

Destaca-se que uma das maiores complexidades existentes no estudo das linhagens das aplicações maliciosas é o uso de atributos ou de características equivalentes , como permissões, e recursos de *hardware* semelhantes. Como apresentado no Experimento 3 do Capítulo 4, permissões utilizadas em jogos também podem ser consideradas perigosas. Outros exemplos são comportamentos de *malware* conhecidos como cavalos-de-tróia, estando em várias outras categorias, como *spyware*, *rootkits* e *ransomware*.

Alguns desafios apontados na filogenia de *malware* foram abordados na Seção 2.8.6. Em relação às considerações do presente trabalho, a partir dos experimentos realizados de agrupamentos para filogenia de variantes de códigos maliciosos, ressalta-se como pontos fortes:

- Simplicidade e rapidez de implementação de algoritmos de agrupamento, conforme discorrido na Seção 2.8. Algoritmos como o k-means são frequentemente utilizados por esse motivo.
- Diversidade de algoritmos de agrupamento disponíveis
- Centenas de medidas de similaridade são formuladas e adaptadas frequentemente.
- Possibilidade de análise gráfica dos dendrogramas para facilitar a detecção de comportamentos anômalos no conjunto de exemplares estudados.

Todavia, no estudos dos relacionamentos hierárquicos e de novas famílias, são destacadas as seguintes deficiências do uso de agrupamentos na filogenia:

- Descompactação e extração de características de códigos ofuscados
- A falta de informações relevantes sobre a criação dos *malware*, como processos de desenvolvimento do código
- Falta de informações sobre estratégias de disseminação e do impacto na separação das famílias
- Falta de clareza conceitual e a falta de um formalismo apropriado e necessário para modelar os relacionamentos filogenéticos
- Descendência de múltiplas linhagens
- Rotulação inadequada e falta de padrões entre as empresas de antivírus
- Informações de datas de liberação das variantes nem sempre estão disponíveis ou corretas, dificultando a determinação de uma linha do tempo
- Compartilhamento de atualizações de funcionalidades de múltiplas variantes
- Construção de árvores filogenéticas com milhares ou milhões de amostras, além de demorada, é inviável computacionalmente, podendo necessitar de *petabytes* de memória RAM.

Portanto, o uso de métodos de agrupamento na filogenia de *malware*, como todo método computacional, apresenta vantagens e desvantagens. Como a categorização de código malicioso pode ter propósitos variados, o uso de agrupamento e de filogenia pode ser viável em algumas situações e inviável em outras. Por exemplo, para desenvolvedores de sistemas operacionais é fundamental saber quais vulnerabilidades estão sendo utilizadas para ataques e invasões. Utilizando essas características para distinção, e não tendo a disposição rótulos de famílias, por exemplo, existe viabilidade no uso de agrupamento.

É interessante destacar que para uma pessoa que tem seu dispositivo infectado, é essencial saber como resolver a situação rapidamente, se seus dados estão protegidos e como não se infectar novamente em vez de apenas saber o nome da família e da variante do *malware*.

Partindo disso, apresentam-se as seguintes recomendações:

- **Como escolher o algoritmo de agrupamento?** Conforme explicado na Fase 3 do Fluxo de Atividades (Seção 4.7), a seleção do método depende da natureza dos dados, do número de dimensões das características, do custo computacional e do tempo disponível para processamento. Além de experimentar algoritmos diferentes, é necessário saber avaliar se o resultado foi adequado para o problema estudado. Para isso podem ser utilizados recursos visuais, como dendrogramas e gráficos de dispersão e índices de validação dos agrupamentos, como por exemplo, o Coeficiente de Correlação Cofenética.

- **A medida de similaridade impacta no resultado do agrupamento a partir das características que foram utilizadas?**Sim. Por exemplo, o Experimento 1 teve como um dos objetivos avaliar o uso de três medidas de similaridade sob o mesmo conjunto de características e o mesmo tipo de dados (*string*). Em relação à natureza do tipo de representação de dados utilizado, em experimentos que utilizaram grafos de fluxo de chamadas acionadas pelos códigos maliciosos, a escolha da medida de similaridade e do algoritmo de agrupamento serão altamente impactantes no resultado. Outros tipos de dados incluem imagens, modelos ocultos de Markov e documentos (Tabela 4.17).

- **A representação de dados influencia em alguma das escolhas dos itens anteriores?**Sim. Isso ocorre porque as medidas de similaridade podem ser desenvolvidas para naturezas específicas, algumas sendo mais indicadas para dados numéricos, outras para texto e assim por diante. Como exemplos, a distância Hamming só funciona com *strings* de mesmo comprimento e a distância Canberra é indicada para agrupamento do tipo *Fuzzy*. Alguns cenários de uso de tipo de dados e medidas foram disponibilizados na Tabela 4.17.

Para concluir, em relação à viabilidade da classificação de *malware* utilizando métodos hierárquicos, através dos experimentos realizados e dos resultados analisados na literatura, verifica-se que, dependendo do propósito do agrupamento, o uso ainda é satisfatório. Ainda que a classificação não-supervisionada possa ser deficiente na distinção correta dos exemplares em famílias, seu uso pode ser indicado para outras situações, como na investigação de *datasets* de amostras com algumas rótulos faltantes ou incorretos.

Pelas pesquisas constantes e recentes na área, devido ao aumento constante do número de *malware* e da evolução das técnicas utilizadas pelos seus criadores, acredita-se que existem várias limitações e desafios a serem pesquisados na filogenia de códigos maliciosos. Partindo da proposta inicial da Filogenia, sugere-se a aplicação dos métodos mais recentes de outras áreas que ainda não foram estudados computacionalmente na área de *malware*, como métodos da Medicina e da Biologia, para a classificação e a linhagem de programas maliciosos.

8 CONSIDERAÇÕES FINAIS

Pesquisas referentes à análise de *malware* são de interesse da comunidade científica, especialmente devido à expressiva quantidade de aplicações infectadas com novas variantes de *malware* focados na plataforma Android ou em outros sistemas, que são disponibilizadas em mercados oficiais e paralelos de aplicativos.

Para auxílio de analistas e profissionais da área de segurança, o uso de técnicas de agrupamento de *malware* é amplo e não tão recente. Todavia, como o tempo e o poder de processamento computacional são recursos finitos, são necessárias pesquisas que auxiliem os acadêmicos e profissionais nessas tarefas. Partindo disso, contribuições da presente tese são apresentadas a seguir.

- No Capítulo 3 foi apresentada a revisão de mais de 80 trabalhos correlatos, com mapeamento de informações referentes à configuração dos experimentos e dos métodos propostos pelos autores;
- A criação do fluxo de atividades básicas para o agrupamento de *malware* (Capítulo 4), com indicações do uso de medidas de similaridade e de técnicas de agrupamento a partir da natureza dos dados e do propósito da análise de amostras;
- Estudo com detalhes da similaridade de famílias: a presente tese apresentou, nos experimentos realizados, análise de famílias, com detalhes de características, relacionamentos e similaridade com outras famílias. Algumas pesquisas da literatura apresentam algumas relações de parentesco entre as famílias de *malware* Android. Porém, apresentam informações sobre poucas famílias, como em Jiang e Zhou (2013), no qual são estudadas apenas as famílias Plankton, DroidKungFu, e AnserverBot. Pesquisas como a de Zhou e Jiang (2012a) disponibilizam informações sobre mais famílias, porém focam nos meios de ataques utilizados, não no estudo da similaridade e de características em comuns entre as famílias. Outros avaliam a distribuição de amostras, variantes e crescimento das famílias ao longo do tempo, como em Suarez-Tangil e Stringhini (2018);
- Além disso, estudos como Deshotels et al. (2014), Zhang et al. (2014), Feng et al. (2016) utilizam técnicas de aprendizado de máquina supervisionado, que apesar da eficácia, necessitam de *datasets* corretamente rotulados. Como mencionado anteriormente, técnicas de aprendizado não-supervisionado, como algoritmos de agrupamento, podem ser aplicados em situações onde existem amostras com rótulos faltantes;
- A partir da revisão do estado da arte e dos resultados dos experimentos realizados, foi possível estabelecer um método adaptado de agrupamento utilizando características exclusivas de famílias de *malware* e medóides. Os experimentos realizados também permitiram analisar as diferenças de usos de medidas de similaridade, tipos de ligação (*linkages*) e de algoritmos de agrupamento;
- No Capítulo 5 foi apresentado o desenvolvimento do *framework* Androidgyny, para análise de amostras, extração de características e classificação de variantes de *malware* Android. Já os resultados da avaliação do Androidgyny e a comparação com o Gefdroid, ferramenta correlata, foram discutidos no Capítulo 6.

8.0.1 Publicações relacionadas à pesquisa

- Pimenta, Thalita Scharr Rodrigues, Fabricio Ceschin, and Andre Gregio. "ANDROIDGYNY: Reviewing clustering techniques for Android malware family classification."**Digital Threats: Research and Practice**, 2023.
- PIMENTA, Thalita Scharr Rodrigues; DOS SANTOS, Rafael Duarte Coelho; GRÉGIO, André. Family Matters: On the Investigation of [Malicious] Mobile Apps Clustering. In: **Computational**

Science and Its Applications–ICCSA 2021: 21st International Conference, Cagliari, Italy, September 13–16, 2021, Proceedings, Part III 21. Springer International Publishing, 2021. p. 79-94..

8.0.2 Trabalhos Futuros

A partir dos resultados obtidos, consideram-se os seguintes trabalhos futuros:

- Automação do modelo desenvolvido para grandes conjuntos de amostras adquiridas incrementalmente a partir de ambientes reais.
- Disponibilização do código-fonte do Androidgyny e dos vetores de características, com dezenas de milhares de amostras, em diretório público.
- Avaliação do impacto das formas de representação de dados utilizadas em análise de *malware* e na classificação de variantes em famílias, utilizando técnicas supervisionadas e não-supervisionadas.

REFERÊNCIAS

A. F. A. Kadir, N. S. e Ghorbani, A. A. (2015). "android botnet: What urls are telling us". Em *9th International Conference on Network and System Security (NSS)*, páginas 78–91. Springer.

A. F. A. Kadir, N. S. e Ghorbani, A. A. (2018). Understanding android financial malware attacks: Taxonomy, characterization, and challenges. *Journal of Cyber Security and Mobility*.

Abbas, O. A. (2008). Comparisons between data clustering algorithms. *International Arab Journal of Information Technology (IAJIT)*, 5(3).

Abdullah, J. e Chanderan, N. (2017). Hierarchical density-based clustering of malware behaviour. *Journal of Telecommunication, Electronic and Computer Engineering*, 9:159–164.

Abusitta, A., Li, M. Q. e Fung, B. C. (2021). Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications*, 59:102828.

Acampora, G., Bernardi, M. L., Cimitile, M., Tortora, G. e Vitiello, A. (2018). A fuzzy clustering-based approach to study malware phylogeny. Em *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, páginas 1–8. IEEE.

Acharya, S., Rawat, U. e Bhatnagar, R. (2022a). A comprehensive review of android security: Threats, vulnerabilities, malware detection, and analysis. *Security and Communication Networks*, 2022.

Acharya, S., Rawat, U. e Bhatnagar, R. (2022b). A low computational cost method for mobile malware detection using transfer learning and familial classification using topic modelling. *Applied Computational Intelligence and Soft Computing*, 2022.

Alam, S., Traore, I. e Sogukpinar, I. (2015). Annotated control flow graph for metamorphic malware detection. *The Computer Journal,*.

Almin, S. B. e Chatterjee, M. (2015). A novel approach to detect android malware. *JProcedia Computer Science v45*.

Alswaina, F. e Elleithy, K. (2020). Android malware family classification and analysis: Current status and future directions. *Electronics*, 9(6):942.

Altaher, A. (2017). An improved android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (ehnfc) and permission-based features. *Neural Computing and Applications, 28(12), 4147-4157*.

Alzarooni, K. (2012). *Malware variant detection*. Tese de doutorado, UCL (University College London).

Anderson, B., Quist, D., Neil, J., Storlie, C. e Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal in computer Virology*, 7(4):247–258.

Andresini, G., Appice, A. e Malerba, D. (2020). Dealing with class imbalance in android malware detection by cascading clustering and classification. *Complex Pattern Mining*.

Annachhatre, C., Austin, T. H. e Stamp, M. (2014). Hidden markov models for malware classification. *Journal of Computer Virology and Hacking Techniques*, 11:59–73.

Appice, A., Andresini, G. e Malerba, D. (2020). Clustering-aided multi-view classification: a case study on android malware detection. *Journal of intelligent information systems*, 55(1):1–26.

Ardimento, P., Bernardi, M. L. e Cimitile, M. (2020). Malware phylogeny analysis using data-aware declarative process mining. Em *2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, páginas 1–8. IEEE.

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

Arshad, S., Shah, M. A., Khan, A. e Ahmed, M. (2016). Android malware detection & protection: a survey. *International Journal of Advanced Computer Science and Applications*.

Askari, S. (2021). Fuzzy c-means clustering algorithm for data with unequal cluster sizes and contaminated with noise and outliers: Review and development. *Expert Systems with Applications*, 165:113856.

Asquith, M. (2016). Extremely scalable storage and clustering of malware metadata. *Journal of Computer Virology and Hacking Techniques*.

Atzeni, A., Díaz, F., Marcelli, A., Sánchez, A., Squillero, G. e Tonda, A. (2018). Countering android malware: A scalable semi-supervised approach for family-signature generation. *IEEE Access*, 6:59540–59556.

Aung, Z. e Zaw, W. (2013). Permission-based android malware detection. *International Journal of Scientific & Technology Research*.

AV-Test (2023). Development of windows malware and pua.

AVTest, I. (2022). Malware. urlhttps://www.av-test.org/en/statistics/malware/.

Awad, R. A. e Sayre, K. D. (2016). Automatic clustering of malware variants. Em *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, páginas 298–303. IEEE.

Babaagba, K. O. e Adesanya, S. O. (2019). A study on the effect of feature selection on malware analysis using machine learning. Em *Proceedings of the 2019 8th international conference on educational and information technology*, páginas 51–55.

Backer, E. (1995). *Computer-assisted reasoning in cluster analysis*. Prentice Hall International (UK) Ltd.

Backes, M., Bugiel, S., Gerling, S. e von Styp-Rekowsky, P. (2014). Android security framework: Extensible multi-layered access control on android. Em *Proceedings of the 30th annual computer security applications conference*, páginas 46–55.

Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F. e Nazario, J. (2007). Automated classification and analysis of internet malware. Em *International Workshop on Recent Advances in Intrusion Detection*, páginas 178–197. Springer.

Barrera, D., Kayacik, H. G., Van Oorschot, P. C. e Somayaji, A. (2010). A methodology for empirical analysis of permission-based security models and its application to android. Em *Proceedings of the 17th ACM conference on Computer and communications security*, páginas 73–84. ACM.

Baskaran, B. e Ralescu, A. L. (2016). A study of android malware detection techniques and machine learning. Em *MAICS*.

Battista, P., Mercaldo, F., Nardone, V., Santone, A. e Visaggio, C. A. (2016). Identification of android malware families with model checking. Em *ICISSP*.

Bernardi, M. L., Cimitile, M. e Mercaldo, F. (2016). Process mining meets malware evolution: a study of the behavior of malicious code. Em *2016 Fourth International Symposium on Computing and Networking*, páginas 616–622. IEEE.

Biggio, B., Bulò, S. R., Pillai, I., Mura, M., Mequanint, E. Z., Pelillo, M. e Roli, F. (2014a). Poisoning complete-linkage hierarchical clustering. Em *S+SSPR*.

Biggio, B., Rieck, K., Ariu, D., Wressnegger, C., Corona, I., Giacinto, G. e Roli, F. (2014b). Poisoning behavioral malware clustering. Em *AISec '14*.

Bisandu, D. B., Prasad, R. e Liman, M. M. (2019). Data clustering using efficient similarity measures. *Journal of Statistics and Management Systems*, 22(5):901–922.

Blum, A. L. e Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1-2):245–271.

Boehmke, B. e Greenwell, B. (2019). *Hands-on machine learning with R*. Chapman and Hall/CRC.

Bontchev, V. (2004). Anti-virus spamming and the virus-naming mess: Part 2. *Virus Bulletin*.

Bora, D. J., Gupta, D. e Kumar, A. (2014). A comparative study between fuzzy clustering algorithm and hard clustering algorithm. *International Journal of Computer Trends and Technology (IJCTT)*.

Bruzzese, D. e Vistocco, D. (2010). Cutting the dendrogram through permutation tests. *Proceedings of COMPSTAT'2010*, páginas 847–854.

Burguera, I., Zurutuza, U. e Nadjm-Tehrani, S. (2011). Crowdroid: behavior-based malware detection system for android. Em *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, páginas 15–26.

Caldas, D. M. (2016). Análise e extração de características estruturais e comportamentais para perfis de malware. Dissertação de Mestrado, Mestrado em Engenharia Elétrica - Universidade de Brasília, Brasília - DF.

Caldiera, V. R. B.-G. e Rombach, H. D. (1994). Goal question metric paradigm. *Encyclopedia of software engineering*, 1(528-532):6.

Canbek, G., Baykal, N. e Sagiroglu, S. (2017). Clustering and visualization of mobile application permissions for end users and malware analysts. Em *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*, páginas 1–10. IEEE.

Canfora, G., Mercaldo, F., Pirozzi, A. e Visaggio, C. A. (2016). How i met your mother? – an empirical study about android malware phylogenesis. Em *In Proceedings of the 13th International Joint Conference on e-Business and Telecommunications*, páginas 310–317. SCITEPRESS-Science and Technology Publications, Lda.

Carpenter, G. A., Grossberg, S. e Rosen, D. B. (1991). Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural networks*.

Carrera, E. e Erdélyi, G. (2004). Digital genome mapping – advanced binary malware analysis. *Proc. Virus Bull. Int. Conf*, páginas 187–207.

Castillo, C., Samani, R. et al. (2014). Mcafee labs threats report. *McAfee Inc., Santa Clara, CA. Available: http://www. mcafee. com/us/resources/reports/rp-quarterlythreat-q1-2014. pdf*.

Cebrián, M., Alfonseca, M. e A.Ortega (2007). The normalized compression distance is resistant to noise. *IEEE Transactions on Information Theory*.

Cha, S.-H. (2007). Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1.

Chakraborty, S., Stokes, J. W., Xiao, L., Zhou, D., Marinescu, M. e Thomas, A. (2017a). Hierarchical learning for automated malware classification. Em *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*, páginas 23–28. IEEE.

Chakraborty, T., Pierazzi, F. e Subrahmanian, V. S. (2017b). Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Transactions on Dependable and Secure Computing*.

Chang, S., Sun, Y., Chuang, W. L., Chen, M. C., Sun, B. e Takahashi, T. (2018). Antsdroid: Using rasmma algorithm to generate malware behavior characteristics of android malware family. Em *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE.

Cheeseman, P. C., Stutz, J. C. et al. (1996). Bayesian classification (autoclass): theory and results. *Advances in knowledge discovery and data mining*, 180:153–180.

Chen, J., Alalfi, M. H., Dean, T. R. e Zou, Y. (2015). Detecting android malware using clone detection. *Journal of Computer Science and Technology*.

Cheng, B., Tong, Q., Wang, J. e Tian, W. (2019). Malware clustering using family dependency graph. *IEEE Access*, 7:72267–72272.

Chitta, R., Jin, R., Havens, T. C. e Jain, A. K. (2011). Approximate kernel k-means: Solution to large scale kernel clustering. Em *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 895–903.

Cho, G., Cho, J., Song, Y. e Kim, H. (2015). An empirical study of click fraud in mobile advertising networks. Em *2015 10th International Conference on Availability, Reliability and Security*, páginas 382–388. IEEE.

Chouchane, M. R. e Lakhotia, A. (2006). Using engine signature to detect metamorphic malware. Em *Proceedings of the 4th ACM workshop on Recurring malcode*, páginas 73–78.

Cimino, M. G., Francesco, N. D., Mercaldo, F., Santone, A. e Vaglini, G. (2020). Model checking for malicious family detection and phylogenetic analysis in mobile environment. *Computers & Security*.

Cimitile, A., Mercaldo, F., Martinelli, F., Nardone, V., Santone, A. e Vaglini, G. (2017). Model checking for mobile android malware evolution. Em *Proceedings of the 5th International FME Workshop on Formal Methods in Software Engineering*, páginas 24–30. IEEE.

Corne, D., Handl, J. e Knowles, J. (2010). *Encyclopedia of Machine Learning*. Springer US".

Cozzi, E., Vervier, P.-A., Dell'Amico, M., Shen, Y., Bilge, L. e Balzarotti, D. (2020). The tangled genealogy of iot malware. Em *Annual Computer Security Applications Conference*, páginas 1–16.

Crussell, J., Gibler, C. e Chen, H. (2013). Andarwin: Scalable detection of semantically similar android applications. Em *European Symposium on Research in Computer Security*, páginas 182–199. Springer.

Cruz, B., Gupta, D., Kapoor, A., Haifei, L., McLean, D., Moreno, F. et al. (2021). Pandemic fears and mobile banking are popular malware targets. *McAfee Inc., Santa Clara, CA. Available: https://www.mcafee.com/content/dam/global/infographics/McAfeeMobileThreatReport2021.pdf*.

Darmetko, C., Jilcott, S. e Everett, J. (2013). Inferring accurate histories of malware evolution from structural evidence. Em *FLAIRS Conference*.

Deng, C., Song, J., Sun, R., Cai, S. e Shi, Y. (2018). Griden: An effective grid-based and density-based spatial clustering algorithm to support parallel computing. *Pattern Recognition Letters*, 109:81–88.

Deshotels, L., Notani, V. e Lakhotia, A. (2014). Droidlegacy: Automated familial classification of android malware. Em *Proceedings of ACM SIGPLAN on program protection and reverse engineering workshop 2014*, página 3. ACM.

Desnos, A. et al. (2013). Androguard-reverse engineering, malware and goodware analysis of android applications. *URL code. google. com/p/androguard*, 153.

Dhalaria, M. e Gandotra, E. (2021). Android malware detection techniques: A literature review. *Recent Patents on Engineering*, 15(2):225–245.

Dini, G., Martinelli, F., Saracino, A. e Sgandurra, D. (2012). Madam: a multi-level anomaly detector for android malware. Em *Computer Network Security: 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2012, St. Petersburg, Russia, October 17-19, 2012. Proceedings 6*, páginas 240–253. Springer.

Du, K.-L. (2010). Clustering: A neural network approach. *Neural networks*, 23(1):89–107.

Dumitras, T. e Neamtiu, I. (2011). A story of provenance and lineage for malware. Em *CSET*. ACM.

Dutta, D. A. K. (2016). Detection of malware and malicious executables using e-birch algorithm. *Journal of Advanced Computer Science and Applications*.

Dutta, M. et al. (2012). Performance analysis of clustering methods for outlier detection. Em *2012 Second International Conference on Advanced Computing & Communication Technologies*, páginas 89–95. IEEE.

Edem, E. I., Benzaid, C., Al-Nemrat, A. e Watters, P. A. (2014). Analysis of malware behaviour: Using data mining clustering techniques to support forensics investigation. *2014 Fifth Cybercrime and Trustworthy Computing Conference*, páginas 54–63.

El Attar, A., Khatoun, R. e Lemercier, M. (2014). A gaussian mixture model for dynamic detection of abnormal behavior in smartphone applications. Em *2014 global information infrastructure and networking symposium (GIIS)*, páginas 1–6. IEEE.

Elhadi, A. A., Maarof, M. A. e Osman, A. H. (2012). Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences*, 9(3):283.

Eres, R., Landau, G. M. e Parida, L. (2003). A combinatorial approach to automatic discovery of cluster-patterns. Em *International Workshop on Algorithms in Bioinformatics*, páginas 139–150. Springer.

Eshghi, S., Khouzani, M., Sarkar, S. e Venkatesh, S. S. (2016). Optimal patching in clustered malware epidemics. *IEEE/ACM Transactions on Networking*, 24:283–298.

Fan, M., Liu, J., Luo, X., Chen, K., Tian, Z., Zheng, Q. e Liu, T. (2018a). Android malware familial classification and representative sample selection via frequent subgraph analysis. Em *IEEE Transactions on Information Forensics and Securit*, páginas 1890–1905. IEEE.

Fan, M., Liu, J., Luo, X., Chen, K., Tian, Z., Zheng, Q. e Liu, T. (2018b). Android malware familial classification and representative sample selection via frequent subgraph analysis. Em *IEEE Transactions on Information Forensics and Securit*, páginas 1890–1905. IEEE.

Fan, M., Luo, X., Liu, J., Nong, C., Zheng, Q. e Liu, T. (2019a). Ctdroid: leveraging a corpus of technical blogs for android malware analysis. *IEEE Transactions on Reliability*, 69(1):124–138.

Fan, M., Luo, X., Liu, J., Wang, M., Nong, C., Zheng, Q. e Liu, T. (2019b). Graph embedding based familial analysis of android malware using unsupervised learning. Em *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, páginas 771–782. IEEE.

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

Faruki, P., Laxmi, V., Gaur, M. S. e Vinod, P. (2012). Mining control flow graph as api call-grams to detect portable executable malware. Em *SIN '12*.

Fauskrud, J. (2019). Hybrid analysis for android malware family classification in a time-aware setting. Dissertação de Mestrado, NTNU.

Feizollah, A., Anuar, N. B. e Salleh, R. (2018a). Evaluation of network traffic analysis using fuzzy c-means clustering algorithm in mobile malware detection. *Advanced Science Letters*, 24(2):929–932.

Feizollah, A., Anuar, N. B. e Salleh, R. (2018b). Evaluation of network traffic analysis using fuzzy c-means clustering algorithm in mobile malware detection. *Advanced Science Letters*.

Feizollah, A., Anuar, N. B., Salleh, R. e Amalina, F. (2014). Comparative study of k-means and mini batch k-means clustering algorithms in android malware detection using network traffic analysis. Em *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*, páginas 193–197. IEEE.

Feizollah, A., Anuar, N. B., Salleh, R. e Wahab, A. W. A. (2015). A review on feature selection in mobile malware detection. *Digital investigation*.

Feng, Y., Bastani, O., Martins, R., Dillig, I. e Anand, S. (2016). Automated synthesis of semantic malware signatures using maximum satisfiability. *arXiv preprint arXiv:1608.06254*.

Ferreira, D. F. (2008). *Estatística multivariada*. Editora Ufla Lavras.

Fiky, A. H. E., Shenawy, A. E. e Madkour, M. A. (2021). Android malware category and family detection and identification using machine learning. *arXiv preprint arXiv:2107.01927*.

Filiol, E. (2006). Malware pattern scanning schemes secure against black-box analysis. *Journal in Computer Virology*, 2(1):35–50.

Fowler, J. E. (2016). Delta encoding of virtual-machine memory in the dynamic analysis of malware. Em *Data Compression Conference (DCC)*, páginas 592–592. IEEE.

Fraley, C. e Raftery, A. E. (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American statistical Association*.

Gajrani, J., Laxmi, V., Tripathi, M., Gaur, M. S., Zemmari, A., Mosbah, M. e Conti, M. (2020). Effectiveness of state-of-the-art dynamic analysis techniques in identifying diverse android malware and future enhancements. Em *Advances in Computers*, volume 119, páginas 73–120. Elsevier.

Gandotra, E., Bansal, D. e Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*.

Garcia, S. e Noe, M. (2011). *Fractal dimension for clustering and unsupervised and supervised feature selection*. Cardiff University.

Gautam, A. e Rahimi, N. (2023). Viability of machine learning in android scareware detection. *Proceedings of 38th International Confer*, 91:19–26.

Gharacheh, M., Derhami, V., Hashemi, S. e Fard, S. M. H. (2016). Detection of metamorphic malware based on hmm: A hierarchical approach. *International Journal of Intelligent Systems and Applications*, 8:18–25.

Ghosh, K. e Mills, J. (2019). Automated construction of malware families. Em *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, páginas 465–474. Springer.

Goldberg, L. A., Goldberg, P. W., Phillips, C. A. e Sorkin, G. B. (1998). Constructing computer virus phylogenies. *Journal of Algorithms*, 26(1):188–208.

Gosain, A. e Dahiya, S. (2016). Performance analysis of various fuzzy clustering algorithms: a review. *Procedia Computer Science*, 79:100–111.

Gržinić, T. e González, E. B. (2022). Methods for automatic malware analysis and classification: a survey. *International Journal of Information and Computer Security*, 17(1-2):179–203.

Guan, Q., Tang, Y. e Liu, X. (2012). A malware homologous analysis method based on sequence of system function.

Guralnik, D. P., Moran, B., Pezeshki, A. e Arslan, O. (2017). Detecting poisoning attacks on hierarchical malware classification systems. Em *Defense + Security*.

Gurrutxaga, I., Arbelaitz, O., Perez, J. M., Muguerza, J., Martin, J. I. e Perona, I. (2008). Evaluation of malware clustering based on its dynamic behaviour. Em *Proceedings of the 7th Australasian Data Mining Conference-Volume 87*, páginas 163–170.

Gutiérrez-Cárdenas, J. M. e Orihuela, L. L. (2016). Filogenia de malware orientada al análisis de librerías.

Halkidi, M., Batistakis, Y. e Vazirgiannis, M. (2001). On clustering validation techniques. *Journal of intelligent information systems*, 17(2):107–145.

Hall, L. O., Ozyurt, I. B. e Bezdek, J. C. (1999). Clustering with a genetically optimized approach. Em *IEEE Transactions on Evolutionary computation*. IEEE.

Hamed, Y. S. I., AbdulKader, S. N. A. e Mostafa, M. S. M. (2019). Mobile malware detection: A survey. *International Journal of Computer Science and Information Security (IJCSIS)*.

Hamid, I. R. A., Khalid, N. S., Abdullah, N. A., Ab Rahman, N. H. e Wen, C. C. (2017). Android malware classification using k-means clustering algorithm. Em *IOP Conference Series: Materials Science and Engineering (Vol. 226, No. 1, p. 012105)*. IOP Publishing.

Han, J., Pei, J. e Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Hancer, E., Xue, B. e Zhang, M. (2020). A survey on feature selection approaches for clustering. *Artificial Intelligence Review*, 53(6):4519–4545.

Hart, P. E., Stork, D. G. e Duda, R. O. (2000). *Pattern classification*. Wiley Hoboken.

Hashimoto, M. e Mori, A. (2008). Diff/ts: A tool for fine-grained structural change analysis. Em *Virus Bulletin Conference*, páginas 279–288. IEEE.

Hassen, M. e Chan, P. K. (2017). Scalable function call graph-based malware classification. Em *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (pp. 239-248)*, páginas 239–248. ACM.

Haubold, B. e Wiehe, T. (2006). Introduction to computational biology. *Basel, Switzerland: Birkhauser*.

Hayes, M., Walenstein, A. e Lakhotia, A. (2008). Evaluation of malware phylogeny modelling systems using automated variant generation. *Journal in Computer Virology*, 5:335–343.

Haykin, C. (2007). *Redes neurais: princípios e prática*. Bookman Editora.

He, G., Xu, B. e Zhu, H. (2018). Appfa: A novel approach to detect malicious android applications on the network. *Security and Communication Networks*.

Hennig, W. (1966). *Phylogenetic Systematics*. University of Illinois Press.

Hsiao, S. W., Sun, Y. S. e Chen, M. C. (2016). Behavior grouping of android malware family. Em *2016 IEEE International Conference on Communications (ICC)*, páginas 1–6. IEEE.

Huang, Z. (1997). A fast clustering algorithm to cluster very large categorical data sets in data mining. *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (SIGMOD-DMKD'97)*.

Hurier, M., Suarez-Tangil, G., Dash, S. K., Bissyandé, T. F., T., Y. L., Klein, J. e Cavallaro, L. (2017). Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. Em *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE.

Huson, D. H., Rupp, R. e Scornavacca, C. (2010). *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press.

Ilham, S., Abderrahim, G. e Abdelhakim, B. A. (2018). Clustering android applications using k-means algorithm using permissions. Em *The Proceedings of the Third International Conference on Smart City Applications*, páginas 678–690. Springer.

Imran, M., Afzal, M. T. e Qadir, M. A. (2017). A comparison of feature extraction techniques for malware analysis. *Turkish Journal of Electrical Engineering & Computer Sciences*, 25(2):1173–1183.

Institute, A.-T. (2020). Security report 2019/2020. Relatório técnico, AV-TEST Institute, Germany.

Iqbal, S., Yasin, A. e Naqash, T. (2018). Android (nougats) security issues and solutions. Em *2018 IEEE International Conference on Applied System Invention (ICASI)*, páginas 1152–1155. IEEE.

Irani, J., Pise, N. e Phatak, M. (2016). Clustering techniques and the similarity measures used in clustering: A survey. *International journal of computer applications*, 134(7):9–14.

Islam, M. R., Tian, R., Batten, L. M. e Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. *J. Netw. Comput. Appl.*, 36:646–656.

Jamrozik, K. e Zeller, A. (2016). Droidmate: a robust and extensible test generator for android. Em *Proceedings of the International Conference on Mobile Software Engineering and Systems*, páginas 293–294.

Jang, J., Brumley, D. e Venkataraman, S. (2010). Bitshred: Fast, scalable malware triage. *Cylab, Carnegie Mellon University, Pittsburgh, PA, Technical Report CMU-Cylab-10*, 22.

Jang, J., Woo, M. e Brumley, D. (2013). Towards automatic software lineage inference. Em *USENIX Security Symposium*.

Jang, J. W., Yun, J., Mohaisen, A., Woo, J. e Kim, H. K. (2016). Detecting and classifying method based on similarity matching of android malware behavior with profile. *SpringerPlus*.

Jaro, M. A. (1995). Probabilistic linkage of large public health data files. *Statistics in medicine*.

Ji, J.-H., Park, S.-H., Woo, G. e Cho, H.-G. (2008). Generating pylogenetic tree of homogeneous source code in a plagiarism detection system. *International Journal of Control Automation and Systems*, 6:809–817.

Jiang, X. e Zhou, Y. (2013). *Case Studies*, páginas 21–29. Springer New York, New York, NY.

Jiang, X., Zhou, Y., Jiang, X. e Zhou, Y. (2013). A survey of android malware. *Android Malware*, páginas 3–20.

Jiang, X. e Zhu, X. (2009). veye: behavioral footprinting for self-propagating worm detection and profiling. *Knowledge and information systems*, páginas 231–262.

Jilcott, S. (2015). Scalable malware forensics using phylogenetic analysis. *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*, páginas 1–6.

Joo, S. e Hwang, C. (2012). Mobile banking vulnerability: Android repackaging threat. *Virus Bulletin, May*.

Jordaney, R. e Nouretdinov, I. (2016). Technical report 2016-1 — royal holloway , university of london misleading metrics : On evaluating machine learning for malware with confidence.

Kadir, A. F. A., Stakhanova, N. e Ghorbani, A. A. (2016). An empirical analysis of android banking malware. Em *Protecting Mobile Networks and Devices*, páginas 223–246. Auerbach Publications.

Kang, B., Kim, T., Kwon, H., Choi, Y. e Im, E. G. (2012). Malware classification method via binary content comparison. Em *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, páginas 316–321.

Kang, Z., Wen, L., Chen, W. e Xu, Z. (2019). Low-rank kernel learning for graph-based clustering. *Knowledge-Based Systems*, 163:510–517.

Karbab, E. B. e Debbabi, M. (2021a). Petadroid: Adaptive android malware detection using deep learning. Em *Detection of Intrusions and Malware, and Vulnerability Assessment: 18th International Conference, DIMVA 2021, Virtual Event, July 14–16, 2021, Proceedings*, páginas 319–340.

Karbab, E. B. e Debbabi, M. (2021b). Resilient and adaptive framework for large scale android malware fingerprinting using deep learning and nlp techniques. *arXiv preprint arXiv:2105.13491*.

Karim, M. E., Walenstein, A., Lakhotia, A. e Parida, L. (2005a). Malware phylogeny generation using permutations of code. *Journal in Computer Virology*.

Karim, M. E., Walenstein, A., Lakhotia, A. e Parida, L. (2005b). Malware phylogeny using maximal pi-patterns. Em *EICAR 2005 Conference: Best Paper Proceedings*, páginas 156–174.

Kasiran, Z., Awang, N. e Rusli, F. N. (2017). Permission based in android malware classification. *DEStech Transactions on Engineering and Technology Research*.

Katebi, M., Rezakhani, A. e Joudaki, S. (2021). Adcas: Adversarial deep clustering of android streams. *Computers & Electrical Engineering*, 95:107443.

Katos, V. (2007). Network intrusion detection: Evaluating cluster, discriminant, and logit analysis. *Information Sciences 177(15)*.

Kaur, P. e Sharma, S. (2014). Google android a mobile platform: A review. *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, páginas 1–5.

Khalilian, M. e Mustapha, N. (2010). Data stream clustering: Challenges and issues. Em *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1.

Khan, J. e Shahzad, S. (2015). Android architecture and related security risks. *Asian J. Technol. Manag. Res.[ISSN: 2249–0892]*, 5:14–18.

Khoda, M., Imam, T., Kamruzzaman, J., Gondal, I. e Rahman, A. (2019). Selective adversarial learning for mobile malware. Em *Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, páginas 272–279. IEEE.

Khomh, F., Yuan, H. e Zou, Y. (2012). Adapting linux for mobile platforms: An empirical study of android. Em *2012 28th IEEE international conference on software maintenance (ICSM)*, páginas 629–632. IEEE.

Khoo, W. M. e P.Lió (2011). Unity in diversity: Phylogenetic-inspired techniques for reverse engineering and detection of malware familie. *2011 First SysSec Workshop*.

Killam, R., Cook, P. e Stakhanova, N. (2016). Android malware classification through analysis of string literals. Em *Workshop Programme*, página 27.

Kim, H. M., Song, H. M., Seo, J. W. e Kim, H. K. (2018). Andro-simnet: Android malware family classification using social network analysis. Em *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, páginas 1–8. IEEE.

Kim, J. e Warnow, T. (1999). Tutorial on phylogenetic tree estimation. *Intelligent Systems for Molecular Biology*.

Kim, T., Hwang, W., Kim, C., Shin, D.-J., Park, K.-W. e Park, K. H. (2015). Malfinder: Accelerated malware classification system through filtering on manycore system. *2015 International Conference on Information Systems Security and Privacy (ICISSP)*, páginas 1–10.

Kim, T., Hwang, W., Park, K.-W. e Park, K. H. (2014). I-filter: Identical structured control flow string filter for accelerated malware variant classification. *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*, páginas 225–231.

Kim, T. e Park, K.-W. (2015). Malcore: Toward a practical malware identification system enhanced with manycore technology. Em *ICISSP*.

Kinable, J. e Kostakis, O. (2011). Malware classification based on call graph clustering. *Journal in computer virology*, páginas 233–245.

Kolbitsch, C., Comparetti, P. M., Kruegel, C., Kirda, E., Zhou, X.-y. e Wang, X. (2009). Effective and efficient malware detection at the end host. Em *USENIX security symposium*, volume 4, páginas 351–366.

Korczynski, D. (2015). Clusthedroid: clustering android malware. Relatório técnico, Royal Holloway, Univ. London, London, UK, Tech. Rep.

Korine, R. e Hendler, D. (2021). Daemon: dataset/platform-agnostic explainable malware classification using multi-stage feature mining. *IEEE Access*, 9:78382–78399.

Krishna, K. e Murty, N. (1999). Genetic k-means algorithms. Em *Transactions on Systems Man And Cybernetics-Part B: Cybernetics*. IEEE.

Kumar, N. S. A. e Kaur, P. (2016). An advanced approach to polymorphic/metamorpic malware detection using hybrid clustering approach. *International Journal of Engineering Development and Research*, 4:1682–1688.

Kumar, S., Janet, B. e Neelakantan, S. (2022). Identification of malware families using stacking of textural features and machine learning. *Expert Systems with Applications*, 208:118073.

Lakhotia, A., Walenstein, A., Miles, C. e Singh, A. (2013). Vilo: a rapid learning nearest-neighbor classifier for malware triage. *Journal of Computer Virology and Hacking Techniques*, 9:109–123.

Lashkari, A. H., Kadir, A. F. A., Taheri, L. e Ghorbani, A. (2018). Toward developing a systematic approach to generate benchmark android malware datasets and classification. *2018 International Carnahan Conference on Security Technology (ICCST)*, páginas 1–7.

Lee, S., Jung, W., Kim, S. e Kim, E. T. (2019). Android malware similarity clustering using method based opcode sequence and jaccard index. Em *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, páginas 178–183. IEEE.

Lee, T., Cho, H., Park, H. e Kwak, J. (2015). Detection of malware propagation in sensor node and botnet group clustering based on e-mail spam analysis. *International Journal of Distributed Sensor Networks*, 11.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Em *Soviet physics doklady*, páginas 707–710. Soviet Union.

Li, P., Liu, L., Gao, D. e Reiter, M. K. (2010). On challenges in evaluating malware clustering. Em *RAID*.

Li, Q., Hu, Q., Qi, Y., Qi, S., Liu, X. e Gao, P. (2021). Semi-supervised two-phase familial analysis of android malware with normalized graph embedding. *Knowledge-Based Systems*, 218:106802.

Li, Y., Jang, J., Hu, X. e Ou, X. (2017a). Android malware clustering through malicious payload mining. Em *International Symposium on Research in Attacks, Intrusions, and Defenses*, páginas 192–214. Springer.

Li, Y., Yang, Z., Guo, Y. e Chen, X. (2017b). Droidbot: a lightweight ui-guided test input generator for android. Em *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, páginas 23–26. IEEE.

Liebergeld, S. e Lange, M. (2013). Android security, pitfalls and lessons learned. Em *Information Sciences and Systems 2013*, páginas 409–417. Springer.

Liles, S., Poremski, E. e Liles, S. (2015). Fusion of malware and weapons taxonomies for analysis. *Journal of Information Warfare)*, páginas 75–83.

Lim, Y.-K., Parambil, S., Kim, C.-G. e Lee, S.-H. (2012). A selective ahead-of-time compiler on android device. Em *2012 International Conference on Information Science and Applications*, páginas 1–6. IEEE.

Lin, Y., Xu, G., Du, C., Xu, G. e Liu, S. (2022). Low-resource malware family detection by cross-family knowledge transfer. *Electronics*, 11(24):4148.

Linden, R. (2009). Técnicas de agrupamento. *Revista de Sistemas de Informação da FSMA*.

Ling, Y. T. e Sani, N. F. M. (2017). Short review on metamorphic malware detection in hidden markov models. *International Journal*, 7(2).

Liu, H. e Motoda, H. (2007). *Computational methods of feature selection*. CRC Press.

Liu, J., Wang, Y., Dai XIE, P. e Wang, Y. J. (2017). Inferring phylogenetic network of malware families based on splits graph. *IEICE TRANSACTIONS on Information and Systems*.

Liu, J., Wang, Y. e Wang, Y. (2016). Inferring phylogenetic networks of malware families from api sequences. Em *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE.

Liu, Z., Wang, R., Japkowicz, N., Tang, D., Zhang, W. e Zhao, J. (2021). Research on unsupervised feature learning for android malware detection based on restricted boltzmann machines. *Future Generation Computer Systems*, 120:91–108.

Lou, S., Cheng, S., Huang, J. e Jiang, F. (2019). Tfdroid: Android malware detection by topics and sensitive data flows using machine learning techniques. Em *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*, páginas 30–36. IEEE.

Ma, J., Dunagan, J., Wang, H. J., Savage, S. e Voelker, G. M. (2006). Finding diversity in remote code injection exploits. Em *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, páginas 53–64.

MacLean, D., Komatineni, S. e Allen, G. (2015). Exploring android persistence and content providers. Em *Pro Android 5*, páginas 559–605. Springer.

Maggi, F., Bellini, A., Salvaneschi, G. e Zanero, S. (2011). Finding non-trivial malware naming inconsistencies. Em *International Conference on Information Systems Security*, páginas 144–159. Springer.

Malhotra, A. e Bajaj, K. (2016). A survey on various malware detection techniques on mobile platform. *Int J Comput Appl*.

Manzano, C., Meneses, C., Leger, P. e Fukuda, H. (2022). An empirical evaluation of supervised learning methods for network malware identification based on feature selection. *Complexity*, 2022.

Martín, A., Lara-Cabrera, R. e Camacho, D. (2019a). Android malware detection through hybrid features fusion and ensemble classifiers: the andropytool framework and the omnidroid dataset. *Information Fusion*, 52:128–142.

Martín, I., Hernández, J. A. e De Los Santos, S. (2019b). Machine-learning based analysis and classification of android malware signatures. *Future Generation Computer Systems*, 97:295–305.

Martinelli, F., Mercaldo, F. e Saracino, A. (2018). Poster: A framework for phylogenetic analysis in mobile environment. Em *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, páginas 825–827. ACM.

Martín, A., Menéndez, H. D. e Camacho, D. (2017). Mocdroid: multi-objective evolutionary classifier for android malware detection. *Soft Computing, 21(24), 7405-7415*.

Mathur, K. e Hiranwal, S. (2013). A survey on techniques in detection and analyzing malware executables. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(4):422–428.

Matsubara, E. T. (2004). *O algoritmo de aprendizado semi-supervisionado co-training e sua aplicação na rotulação de documentos*. Tese de doutorado, Universidade de São Paulo.

Mawlood-Yunis, A.-R. (2022). Your first android application. Em *Android for Java Programmers*, páginas 95–133. Springer.

Medina, L. E. G. (2016). Droidkungfu and the exploits rageagainstthecage and exploid for android.

Meng, G. (2017). *A Semantic-based Analysis of Android Malware for Detection, Generation, and Trend Analysis*. Tese de doutorado, QuNanyang Technological University.

Metz, J. (2006). Análise e extração de características estruturais e comportamentais para perfis de malware. Dissertação de Mestrado, Mestra em Ciências de Computação e Matemática Computacional - USP., São Carlos - SP.

Milosevic, N., Dehghantanha, A. e Choo, K. K. R. (2017). Machine learning aided android malware classification. *Computers & Electrical Engineering*.

Ming, J., Xin, Z., Lan, P., Wu, D., Liu, P. e Mao, B. (2016). Impeding behavior-based malware analysis via replacement attacks to malware specifications. *Journal of Computer Virology and Hacking Techniques*, 13:193–207.

Mirzaei, O., Suarez-Tangil, G., de Fuentes, J. M., Tapiador, J. e Stringhini, G. (2019). Andrensemble: Leveraging api ensembles to characterize android malware families. Em *Proceedings of the 2019 ACM Asia conference on computer and communications security*, páginas 307–314.

Mohini, T., Kumar, S. A. e Nitesh, G. (2013). Review on android and smartphone security. *Research Journal of Computer and Information Technology Science*.

Mohite, S. e Sonar, P. (2014). A survey on mobile malware: A war without end. *International Journal of Computer Science and Business Informatics*, 9(1):23–35.

Moser, A., Kruegel, C. e Kirda, E. (2007). Limits of static analysis for malware detection. Em *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, páginas 421–430. IEEE.

Moses, A. e Morris, S. (2021). Analysis of mobile malware: a systematic review of evolution and infection strategies. *Journal of Information Security and Cybercrimes Research*, 4(2):103–131.

Mosharrat, N., Sarker, I. H., Anwar, M. M., Islam, M. N., Watters, P. e Hammoudeh, M. (2022). Automatic malware categorization based on k-means clustering technique. Em *Proceedings of the International Conference on Big Data, IoT, and Machine Learning: BIM 2021*, páginas 653–664. Springer.

Moubarak, J., Chamoun, M. e Filiol, E. (2017). Comparative study of recent mea malware phylogeny. *2017 2nd International Conference on Computer and Communication Systems (ICCCS)*, páginas 16–20.

Naeem, H., Guo, B., Ullah, F. e Naeem, M. R. (2019). A cross-platform malware variant classification based on image representation. *KSII Transactions on Internet & Information Systems*.

Naik, S. e Dessai, A. (2021). Malware classification approaches using machine learning techniques: A review. Em *2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)*, páginas 111–117. IEEE.

Nakazato, J., Song, J., Eto, M., Inoue, D. e Nakao, K. (2011). A novel malware clustering method using frequency of function call traces in parallel threads. *IEICE transactions on information and systems*, 94(11):2150–2158.

Narayanan, A., Soh, C., Chen, L., Liu, Y. e Wang, L. (2018). apk2vec: Semi-supervised multi-view representation learning for profiling android applications. Em *2018 IEEE International Conference on Data Mining (ICDM)*, páginas 357–366. IEEE.

Narra, U., Troia, F. D., Visaggio, C. A., Austin, T. H. e Stamp, M. (2015). Clustering versus svm for malware detection. *Journal of Computer Virology and Hacking Techniques*, 12:213–224.

Nasıbov, E. e Kandemır-Cavas, C. (2011). Owa-based linkage method in hierarchical clustering: Application on phylogenetic trees. *Expert Systems with Applications*, 38(10):12684–12690.

Nerurkar, P., Chandane, A. S. M. e Bhirud, S. (2018). A novel heuristic for evolutionary clustering. Em *Procedia Computer Science*, páginas 780–789. ScienceDirect.

Neugschwandtner, M., Comparetti, P. M., Jacob, G. e Krügel, C. (2011). Forecast: skimming off the malware cream. Em *ACSAC '11*.

Nomura, K., Chiba, D., Akiyama, M. e Uchida, M. (2021). Auto-creation of android malware family tree. Em *ICC 2021-IEEE International Conference on Communications*, páginas 1–6. IEEE.

Nopiah, Z., Khairir, M., Abdullah, S., Baharin, M. e Arifin, A. (2010). Time complexity analysis of the genetic algorithm clustering method. Em *Proceedings of the 9th WSEAS international conference on signal processing, robotics and automation, ISPRA*, volume 10, páginas 171–176.

Odusami, M., Abayomi-Alli, O., Misra, S., Shobayo, O., Damasevicius, R. e Maskeliunas, R. (2018). Android malware detection: A survey. Em *International Conference on Applied Informatics*, páginas 255–266. Springer.

OHA (2007). Open handset alliance.

Oprisa, C., Cabau, G. e Sebestyen, G. (2014). Malware clustering using suffix trees. *Journal of Computer Virology and Hacking Techniques*, 12:1–10.

Otani, Y., Ueno, A. e Takubo, T. (2020). Android malware classification using flattened image representation and senet. *ICIC express letters. Part B, Applications: an international journal of research and surveys*, 11(8):761–766.

Oyen, D., Anderson, B., Sentz, K. e Anderson-Cook, C. (2017). Order priors for bayesian network discovery with an application to malware phylogeny. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 10(5):343–358.

Pai, S. (2015). A comparison of clustering techniques for malware analysis.

Pai, S., Troia, F. D., Visaggio, C. A., Austin, T. H. e Stamp, M. (2016). Clustering for malware classification. *Journal of Computer Virology and Hacking Techniques*, 13:95–107.

Pandit, S. e Gupta, S. (2011). A comparative study on distance measuring approaches for clustering. *International Journal of Research in Computer Science, 2(1)*.

Pandit, S., Gupta, S. et al. (2011). A comparative study on distance measuring approaches for clustering. *International journal of research in computer science*, 2(1):29–31.

Park, S., Suresh, N. C. e Jeong, B.-K. (2008). Sequence-based clustering for web usage mining: A new experimental framework and ann-enhanced k-means algorithm. *Data & Knowledge Engineering*, 65(3):512–543.

Park, Y., Reeves, D. S. e Stamp, M. (2013). Deriving common malware behavior through graph clustering. *Comput. Secur.*, 39:419–430.

Pavlenko, E. Y., Yarmak, A. V. e Moskvin, D. A. (2017). Application of clustering methods for analyzing the security of android applications. *Automatic Control and Computer Sciences*.

Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J. e Cavallaro, L. (2019). {TESSERACT}: Eliminating experimental bias in malware classification across space and time. Em *28th USENIX Security Symposium (USENIX Security 19)*, páginas 729–746.

Perdisci, R., Ariu, D. e Giacinto, G. (2013). Scalable fine-grained behavioral clustering of http-based malware. *Comput. Networks*, 57:487–500.

Perdisci, R. e U, M. (2012). Vamo: towards a fully automated malware clustering validity analysis. Em *Proceedings of the 28th Annual Computer Security Applications Conference*, páginas 329–338.

Perdisci, R., W.Lee e Feamster, N. (2010). Behavioral clustering of http-based malware and signature generation using malicious network traces. Em *NSDI,Vol. 10*, página 14.

P.Faruki, V.Laxmi, A.Bharmal, M.S.Gaur e V.Ganmoor (2015). Androsimilar: Robust signature for detecting variants of android malware. *Journal of Information Security and Applications*.

Pfeffer, A., Call, C., Chamberlain, J., Kellogg, L., Ouellette, J., Patten, T. e Hall, R. (2012). Malware analysis and attribution using genetic information. Em *2012 7th International Conference on Malicious and Unwanted Software*, páginas 39–45. IEEE.

Pfitzner, D., Leibbrandt, R. e Powers, D. (2009). Characterization and evaluation of similarity measures for pairs of clusterings. *Knowledge and Information Systems*, 19:361–394.

Phuong, T. M., Lin, Z. e Altman, R. B. (2005). Choosing snps using feature selection. Em *2005 IEEE Computational Systems Bioinformatics Conference (CSB'05)*, páginas 301–309. IEEE.

Pieterse, H. e Burke, I. (2015a). Evolution study of android botnets. Em *Iccws 2015—The proceedings of the 10th international conference on cyber warfare and security: ICCWS2015*, página 232.

Pieterse, H. e Burke, I. (2015b). Evolution study of android botnets. Em *Iccws 2015—The proceedings of the 10th international conference on cyber warfare and security*, página 232.

Pitolli, G., Laurenza, G., Aniello, L., Querzoni, L. e Baldoni, R. (2021). Malfamaware: automatic family identification and malware classification through online clustering. *International Journal of Information Security*, 20(3):371–386.

Raff, E. e Nicholas, C. K. (2017). Malware classification and class imbalance via stochastic hashed lzjd. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*.

Ragonnet-Cronin, M., Hodcroft, E., Hué, S., Fearnhill, E., Delpech, V., Brown, A. J. L. e Lycett, S. (2013). Automated analysis of phylogenetic clusters. *BMC bioinformatics*, 14(1):1–10.

Randhawa, T. S. (2022). Security and trust. Em *Mobile Applications*, páginas 571–630. Springer.

Rashed, M. e Suarez-Tangil, G. (2021). An analysis of android malware classification services. *Sensors*, 21(16):5671.

Rathore, H., Sahay, S. K., Chaturvedi, P. e Sewak, M. (2018). Android malicious application classification using clustering. Em *International Conference on Intelligent Systems Design and Applications*, páginas 659–667. Springer.

Rathore, H., Sahay, S. K., Thukral, S. e Sewak, M. (2021). Detection of malicious android applications: Classical machine learning vs. deep neural network integrated with clustering. Em *Broadband Communications, Networks, and Systems: 11th EAI International Conference, BROADNETS 2020, Qingdao, China, December 11–12, 2020, Proceedings 11*, páginas 109–128. Springer.

Ravi, S., Balakrishnan, N. e Venkatesh, B. (2013). Behavior-based malware analysis using profile hidden markov models. *2013 International Conference on Security and Cryptography (SECRYPT)*, páginas 1–12.

Reddy, D. e Pujari, A. K. (2006). N-gram analysis for computer virus detection. *Journal in Computer Virology*, 2(3):231–239.

Riasat, R., Sakeena, M., Chong, W. A. N. G., Sadiq, A. H. e Wang, Y. J. (2016). A survey on android malware detection techniques. Em *DEStech Transactions on Computer Science and Engineering*. wcne.

Rieck, K., Trinius, P., Willems, C. e Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668.

Rokach, L. e Maimon, O. (2005). Clustering methods. Em *Data mining and knowledge discovery handbook*, páginas 321–352. Springer.

S. Josh, R. K. e Joshi, L. K. (2015). Android botnet: An upcoming challenge. Em *In National Conference on Advances in Engineering, Technology & Management*, páginas 5–10.

Salvador, S. e Chan, P. (2004). Determining the number of clusters/segments in hierarchical clustering/-segmentation algorithms. Em *16th IEEE international conference on tools with artificial intelligence*, páginas 576–584. IEEE.

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

Sanz, B., Santos, I., Ugarte-Pedrero, X., Laorden, C., Nieves, J. e Bringas, P. (2014). Anomaly detection using string analysis for android malware detection. Em *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, páginas 469–478. Springer.

Sarraille, J. e DiFalco, P. (2007). Fd3.

Sartea, R., Preda, M. D., Farinelli, A., Giacobazzi, R. e Mastroeni, I. (2016). Active android malware analysis: an approach based on stochastic games. Em *Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering*, páginas 1–10.

Saudi, M. M., Sukardi, S., Syafiq, A. S. M., Ahmad, A. e Husainiamer, M. (2019). Mobile malware classification for cyber physical system (cps) based on phylogenetics. *International Journal of Engineering and Advanced Technology (IJEAT)*.

Schmidt, A. D., Bye, R., Schmidt, H. G., Clausen, J., Kiraz, O., Yuksel, K. A., Camtepe, S. A. e Albayrak, S. (2009). Static analysis of executables for collaborative malware detection on android. *2009 IEEE International Conference on Communications*.

Security, H. N. (2013). Android malware continues to rise.

Seideman, J. D., Khan, B. e Vargas, A. C. (2014). Identifying malware genera using the jensen-shannon distance between system call traces. *Malicious and Unwanted Software: The Americas (MALWARE), 2014 9th International Conference*.

Shakya, S. e Dave, M. (2022). Analysis, detection, and classification of android malware using system calls. *arXiv preprint arXiv:2208.06130*.

Shalaginov, A., Banin, S., Dehghantanha, A. e Franke, K. (2018). Machine learning aided static malware analysis: A survey and tutorial. *Cyber threat intelligence*, páginas 7–45.

Shang, F., Y.Li, Xiaolin, D. e He, D. (2018). Android malware detection method based on naive bayes and permission correlation algorithm. *Cluster Computing*.

Shao, Y., Luo, X., Qian, C., Zhu, P. e Zhang, L. (2014). Towards a scalable resource-driven approach for detecting repackaged android applications. Em *Proceedings of the 30th Annual Computer Security Applications Conference*, páginas 56–65. ACM.

Sharma, S., Krishna, C. R. e Kumar, R. (2021). Ransomdroid: Forensic analysis and detection of android ransomware using unsupervised machine learning technique. *Forensic Science International: Digital Investigation*, 37:301168.

Sharma, S., Kumar, R. e Krishna, C. R. (2020). Ransomanalysis: The evolution and investigation of android ransomware. Em *Proceedings of International Conference on IoT Inclusive Life (ICIIL 2019), NITTTR Chandigarh, India*, páginas 33–41. Springer.

Shawe-Taylor, J. e Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge University Press.

Shi, H., Hamagami, T., Yoshioka, K., Xu, H., Tobe, K. e Goto, S. (2014). Structural classification and similarity measurement of malware. *IEEJ Transactions on Electrical and Electronic Engineering*, 9.

Siddiqui, M., Wang, M. C. e Lee, J. (2008). A survey of data mining techniques for malware detection using file features. Em *Proceedings of the 46th annual southeast regional conference on xx*, páginas 509–510.

Sihag, V., Vardhan, M. e Singh, P. (2021). A survey of android application and malware hardening. *Computer Science Review*, 39:100365.

Sikorski, M. e Honig, A. (2012). *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press.

Silva, P., Akhavan-Masouleh, S. e Li, L. (2018). Improving malware detection accuracy by extracting icon information. *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, páginas 408–411.

Singh, A., Arora, R. e Pareek, H. (2017). Malware analysis using multiple api sequence mining control flow graph. *ArXiv*, abs/1707.02691.

Singh, A., Walenstein, A. e Lakhotia, A. (2012). Tracking concept drift in malware families. Em *AISec '12*.

Singla, S., Gandotra, E., Bansal, D. e Sofat, S. (2015). Detecting and classifying morphed malwares: A survey. *International Journal of Computer Applications*, 122(10).

Sisaat, K., Kittitornkun, S., Kikuchi, H., Yukonhiatou, C., Terada, M. e Ishii, H. (2017). A spatio-temporal malware and country clustering algorithm: 2012 iij mitf case study. *International Journal of Information Security*, páginas 459–473.

Skovoroda, A. e Gamayunov, D. (2015). Review of the mobile malware detection approaches. *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*.

Sokal, R. R. e Rohlf, F. J. (1962). The comparison of dendrograms by objective methods. *Taxon*, páginas 33–40.

Spreitzenbarth, M. e Freiling, F. (2012). Android malware on the rise.

Statcounter (2023). Operating system market share worldwide.

Struyf, A., Hubert, M. e Rousseeuw, P. (1997). Clustering in an object-oriented environment. *Journal of Statistical Software*, 1:1–30.

Suarez-Tangil, G. e Stringhini, G. (2018). Eight years of rider measurement in the android malware ecosystem: evolution and lessons learned. *arXiv preprint arXiv:1801.08115*.

Suarez-Tangil, G., Tapiador, J. E., Peris-López, P. e Alís, J. B. (2014a). Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Syst. Appl.*, 41:1104–1117.

Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P. e Blasco, J. (2014b). Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications*, 41(4):1104–1117.

Suyal, H., Panwar, A. e Negi, A. S. (2014). Text clustering algorithms: A review. *International Journal of Computer Applications*, 96(24).

Swarndeep Saket, J. e Pandya, S. (2016). An overview of partitioning algorithms in clustering techniques. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 5(6):1943–1946.

Syverson, P., Dingledine, R. e Mathewson, N. (2004). Tor: The secondgeneration onion router. Em *Usenix Security*, páginas 303–320.

Taha, A. A. e Malebary, S. J. (2021). Hybrid classification of android malware based on fuzzy clustering and the gradient boosting machine. *Neural Computing and Applications*, 33(12):6721–6732.

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

Tam, K., Feizollah, A., Anuar, N. B., Salleh, R. e Cavallaro, L. (2017). The evolution of android malware and android analysis techniques. Em *ACM Computing Survey*. ACM.

Tan, P.-N., Steinbach, M. e Kumar, V. (2006). Data mining introduction. *People's Posts and Telecommunications Publishing House, Beijing*.

Ullah, F., Srivastava, G. e Ullah, S. (2022). A malware detection system using a hybrid approach of multi-heads attention-based control flow traces and image visualization. *Journal of Cloud Computing*, 11(1):1–21.

Vega Vega, R., Quintian, H., Cambra, C., Basurto, N., Herrero, A. e Calvo-Rolle, J. L. (2019). Delving into android malware families with a novel neural projection method. *Complexity*.

Verma, S. e Muttoo, S. (2016). An android malware detection framework-based on permissions and intents. *Defence Science Journal*.

Vicini, L. (2005). Análise multivariada: da teoria à prática.

Vidas, T., Tan, J., Nahata, J., Tan, C. L., Christin, N. e Tague, P. (2014). A5: Automated analysis of adversarial android applications. Em *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, páginas 39–50.

Vinod, P., Laxmi, V., Gaur, M. S. e Chauhan, G. (2012). Momentum: metamorphic malware exploration techniques using msa signatures. *International Conference on Innovations in Information Technology (IIT)*, páginas 232–237.

Vlachos, V., Ilioudis, C. e Papanikolaou, A. (2011). On the evolution of malware species. Em *ICGS3/e-Democracy*.

Wagener, G., Dulaunoy, A. et al. (2008). Malware behaviour analysis. *Journal in computer virology*, 4(4):279–287.

Walenstein, A. e Lakhotia, A. (2012a). A transformation-based model of malware derivation. Em *2012 7th International Conference on Malicious and Unwanted Software*, páginas 17–25. IEEE.

Walenstein, A. e Lakhotia, A. (2012b). A transformation-based model of malware derivation. Em *7th International Conference on Malicious and Unwanted Software*, páginas 17–25. IEEE.

Wang, C. (2006). Malware detection. advances in information security.

Wang, L., He, R., Wang, H., Xia, P., Li, Y., Wu, L., Zhou, Y., Luo, X., Sui, Y., Guo, Y. et al. (2021). Beyond the virus: a first look at coronavirus-themed android malware. *Empirical Software Engineering*, 26(4):82.

Wang, S., Chen, Z., Li, X., Wang, L., Ji, K. e Zhao, C. (2017). Android malware clustering analysis on network-level behavior. Em *International Conference on Intelligent Computing*, páginas 796–807. Springer.

Wang, S., Tang, J. e Liu, H. (2015). Embedded unsupervised feature selection. Em *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Wehner, S. (2007). Analyzing worms and network traffic using compression. *Journal of Computer Security*, 15(3):303–320.

Wei, F., Li, Y., Roy, S., Ou, X. e Zhou, W. (2017). Deep ground truth analysis of current android malware. Em *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, páginas 252–276. Springer.

Weichselbaum, L., Neugschwandtner, M., Lindorfer, M., Fratantonio, Y., van der Veen, V. e Platzer, C. (2014). A practical test for univariate and multivariate normality. Relatório técnico, Vienna University of Technology, Tech. Rep.

Wicherski, G. (2009). pehash: A novel approach to fast malware clustering. *LEET*, 9:8.

Willems, C., Holz, T. e Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5(2):32–39.

Winkler, W. E. (1999). The state of record linkage and current research problems. *Statistics of Income Division - Internal Revenue Service Publication*.

Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M. e Wu, K. P. (2012). Droidmat: Android malware detection through manifest and api calls tracing. Em *2012 Seventh Asia Joint Conference on Information Security*, páginas 62–69. IEEE.

Wüchner, T., Ochoa, M. e Pretschner, A. (2014). Malware detection with quantitative data flow graphs. *Proceedings of the 9th ACM symposium on Information, computer and communications security*.

Xiao, J., Han, Q. e Gao, Y. (2021). Hybrid classification and clustering algorithm on recent android malware detection. Em *2021 5th International Conference on Computer Science and Artificial Intelligence*, páginas 249–255.

Xie, N., Wang, X., Wang, W. e Liu, J. (2019). Fingerprinting android malware families. *Frontiers of Computer Science*, 13:637–646.

Xiong, Z., Guo, T., Zhang, Q., Cheng, Y. e Xu, K. (2018). Android malware detection methods based on the combination of clustering and classification. Em *International Conference on Network and System Security*, páginas 411–422. Springer.

Xu, D. e Tian, Y. (2015). A comprehensive survey of clustering algorithms. Em *Annals of Data Science*, páginas 165–193. Springer.

Xu, J., Li, Y., Deng, R. e Xu, K. (2020). Sdac: A slow-aging solution for android malware detection using semantic distance based api clustering. *IEEE Transactions on Dependable and Secure Computing*.

Xu, R. e Donald Wunsch, I. I. (2005). Survey of clustering algorithms. Em *IEEE TRANSACTIONS ON NEURAL NETWORKS*. IEEE.

Xu, X., Ester, M., Kriegel, H. P. e Sander, J. (1998). A distribution-based clustering algorithm for mining in large spatial databases. Em *Proceedings 14th International Conference on Data Engineering*. IEEE.

Y. Kim, K. J. L. e Chan, C. C. (2016). Using droiddream android malware behavior for identification of other android malware families. Em *In Proceedings of the International Conference on Security and Management (SAM)*. Athens: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Yadav, N., Sharma, A. e Doegar, A. (2016). A survey on android malware detection. *International Journal of New Technology and Research*.

Yan, P. e Yan, Z. (2018). A survey on dynamic mobile malware detection. *Software Quality Journal*.

Yang, W., Kong, D., Xie, T. e Gunter, C. A. (2017). Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. Em *Proceedings of the 33rd Annual Computer Security Applications Conference*, páginas 288–302. ACM.

Ye, Y., Li, T., Adjeroh, D. A. e Iyengar, S. S. (2017). A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50:1 – 40.

Ye, Y., Li, T., Chen, Y. e Jiang, Q. (2010a). Automatic malware categorization using cluster ensemble. Em *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 95–104.

Ye, Y., Li, T., Huang, K., Jiang, Q. e Chen, Y. (1983). Parsimony in systematics: biological and statistical issues. *Annual review of ecology and systematics*.

Ye, Y., Li, T., Huang, K., Jiang, Q. e Chen, Y. (2010b). Hierarchical associative classifier (hac) for malware detection from the large and imbalanced gray list. *Journal of Intelligent Information Systems*.

Yim, O. e Ramdeen, K. T. (2015). Hierarchical cluster analysis: comparison of three linkage measures and application to psychological data. *The quantitative methods for psychology*, 11(1):8–21.

Yu, B., Fang, Y., Yang, Q., Tang, Y. e Liu, L. (2018). A survey of malware behavior description and analysis. *Frontiers of Information Technology & Electronic Engineering*.

Yu, R. (2013). Ginmaster: a case study in android malware. Em *Virus bulletin conference*, páginas 92–104.

Yuan, L.-P., Hu, W., Yu, T., Liu, P. e Zhu, S. (2019). Towards large-scale hunting for android negative-day malware. Em *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, páginas 533–545.

Zachariah, R., Akash, K., Yousef, M. S. e Chacko, A. M. (2017). Android malware detection a survey. Em *2017 IEEE international conference on circuits and systems (ICCS)*, páginas 238–244. IEEE.

Zaïane, O. R., Foss, A., Lee, C. H. e Wang, W. (2002). On data clustering analysis: Scalability, constraints, and validation. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*.

Zhang, M., Duan, Y., Yin, H. e Zhao, Z. (2014). Semantics-aware android malware classification using weighted contextual api dependency graphs. Em *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, páginas 1105–1116. ACM.

Zhang, Y., Ren, W., Zhu, T. e Ren, Y. (2019a). Saas: A situational awareness and analysis system for massive android malware detection. *Future Generation Computer Systems*, 95:548–559.

Zhang, Y., Sui, Y., Pan, S., Zheng, Z., Ning, B., Tsang, I. e Zhou, W. (2019b). Familial clustering for weakly-labeled android malware using hybrid representation learning. Em *IEEE Transactions on Information Forensics and Security*. IEEE.

Zhao, L., Wang, J., Chen, Y., Wu, F., Liu, Y. et al. (2021). Famdroid: learning-based android malware family classification using static analysis. *arXiv preprint arXiv:2101.03965*.

Zhou, Y. e Jiang, X. (2011). An analysis of the anserverbot trojan. Relatório técnico, Dept. Comput. Sci., North Carolina State Univ.

Zhou, Y. e Jiang, X. (2012a). Dissecting android malware: Characterization and evolution. Em *2012 IEEE symposium on security and privacy*, páginas 95–109. IEEE.

Zhou, Y. e Jiang, X. (2012b). Dissecting android malware: Characterization and evolution. Em *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, páginas 95 – 109. IEEE.

Zhu, X., Li, Y., Wang, J., Zheng, T. e Fu, J. (2020). Automatic recommendation of a distance measure for clustering algorithms. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(1):1–22.

8.1 GLOSSÁRIO

- **Adware** - Um tipo de software indesejado que causa excesso de anúncios e janelas pop-ups no dispositivo ou no navegador *Web*.
- **Agrupamento ou *Clustering*** - Técnica de aprendizado não-supervisionado, utilizada para encontrar padrões em conjuntos de dados.
- ***Backdoor*** - São tipos de *trojans* que permitem acesso ao sistema infectado e seu controle remoto.
- **Boot** - Termo que se refere a evento de inicialização do sistema/dispositivo.
- ***C& C Server*** - Significa *Command-and-control server*, sendo um servidor remoto utilizado para ataques, transmitindo instruções para dispositivos infectados.
- ***Cluster*** - Grupo de elementos parecidos, unidade elementar de técnicas de agrupamento.
- ***Dataset*** - Conjunto de elementos, nesta pesquisa se rerere a conjunto de amostras maliciosas ou não.
- **Família** - Grupo de classificação de *malware* com comportamentos e características similares.
- ***Goodware*** - Termo utilizado em Segurança Computacional para amostras não-maliciosas.
- ***Ground-truth*** - Conjunto de informações que se sabe que são verdadeiras, utilizado para avaliação de técnicas de classificação, por exemplo.
- ***Honeypot*** - Técnica de emboscada utilizada para atrair atacantes em ambientes programados.
- ***Host*** - Ponto conectado à rede de computadores, podendo ser dispositivos como celulares com acesso à internet.
- ***QRCode*** - Espécie de código de barras em formato específico de quadrado.
- *Malware* - *Software* malicioso utilizado para compremeter sistemas e dispositivos.
- ***Outlier*** - Anomalias em conjuntos são chamadas de *outliers*.
- ***Overfitting*** - Termo usado para descrever quando um modelo se ajusta bem a um conjunto de treinamento, mas se mostra ineficaz para dados em que o modelo não foi treinado anteriormente.
- **PUA** - Sigla para *Potentially unwanted applications*, sendo um termo genérico para aplicações maliciosas.
- ***Scareware*** - Tipo de *software* malicioso que engana os usuários, levando-os a comprar aplicativos para problemas que geralmente não existem em seus dispositivos.
- ***Script*** - Sequência de instruções que deve ser executada de modo ordenado, sendo similar ao conceito de algoritmo.
- **SMS** - A sigla de *Short Message Service*, tipo de serviço utilizando em telefones por meio de mensagens curtas de texto.
- ***Spyware*** - Termo em inglês para *software* espião, coleta informações e atividades do usuário.
- **Variante** - Assim como variantes de vírus biológicos, uma variante de *malware* se refere a uma nova versão de um código malicioso que é criada a partir de modificações de *malware* existentes.