Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Segurança em aplicações Android: uma comparação de ferramentas de análise estática

Autores: Kleidson Alves Corrêa e Lucas Rodrigues Fonseca
Orientadora: Profa. Dra. Elaine Venson

Brasília, DF
2023

Kleidson Alves Corrêa e Lucas Rodrigues Fonseca

Segurança em aplicações Android: uma comparação de ferramentas de análise estática

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientadora: Profa. Dra. Elaine Venson

Brasília, DF

2023

Kleidson Alves Corrêa e Lucas Rodrigues Fonseca

Segurança em aplicações Android: uma comparação de ferramentas de análise estática/ Kleidson Alves Corrêa e Lucas Rodrigues Fonseca. – Brasília, DF, 2023-

80 p. : il. (algumas color.) ; 30 cm.

Orientadora: Profa. Dra. Elaine Venson

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Segurança em aplicações Android. 2. Análise estática. I. Profa. Dra. Elaine Venson. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Segurança em aplicações Android: uma comparação de ferramentas de análise estática

CDU 02:141:005.6

Kleidson Alves Corrêa e Lucas Rodrigues Fonseca

Segurança em aplicações Android: uma comparação de ferramentas de análise estática

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 18 de dezembro de 2023:

Profa. Dra. Elaine Venson
Orientadora

Profa. Mestre Cristiane Ramos
Convidado 1

Bacharel Indiana Belianka
Convidado 2

Brasília, DF
2023

Resumo

O número de aplicações para dispositivos móveis cresce constantemente e atinge as mais diversas áreas, tais como entretenimento, saúde e mercado financeiro. Por conta disso, o uso de aplicações nesses dispositivos se tornou muito comum. Elas podem ser utilizadas para realizar um grande número de tarefas do dia a dia, inclusive aquelas com alto grau de criticidade. Com isso, destaca-se a importância da proteção desses aplicativos contra ataques maliciosos. Para auxiliar nessa tarefa, durante o ciclo de desenvolvimento de software, recomenda-se a realização de testes estáticos de segurança, como a análise estática, a qual pode ser otimizada por meio do uso de ferramentas. Diante disso, o objetivo deste trabalho é apresentar um comparativo entre algumas das principais ferramentas gratuitas de análise estática de código quanto a sua capacidade de detecção das principais vulnerabilidades em aplicações móveis de acordo com o Mobile Top 10 do projeto OWASP. Para isso, realiza-se o mapeamento dos resultados obtidos com a execução das ferramentas em uma amostra de aplicações Android para as vulnerabilidades listadas pelo projeto OWASP. Ao final, é possível compreender quais procedimentos foram adotados para a realização do estudo, desde a execução dos testes até a comparação das ferramentas. Também são observados os pontos positivos e negativos das ferramentas, as quais são comparadas sob diferentes perspectivas, incluindo a precisão na detecção das vulnerabilidades com relação aos falsos positivos, tempo levado para concluir os testes e capacidade de detectar diversos tipos de vulnerabilidades do OWASP Mobile Top 10. Além disso, são listadas as vulnerabilidades que foram mais identificadas no contexto de aplicações móveis.

Palavras-chave: aplicações móveis; segurança de software; análise estática de código; vulnerabilidades; OWASP.

Abstract

The number of applications for mobile devices is constantly growing and reaches various areas, such as entertainment, healthcare and the financial market. As a result, the use of applications on these devices has become very common. They can be used to perform a wide range of everyday tasks, including those with a high degree of criticality. This highlights the importance of protecting these applications against malicious attacks. To assist in this task, during the software development cycle, it is recommended to carry out static security testing, such as static analysis, which can be optimized through the use of tools. Therefore, the goal is to present a comparison between some of the main free static code analysis tools regarding their ability to detect the main vulnerabilities in mobile applications according to the Mobile Top 10 of the OWASP project. For this purpose, the results obtained with the execution of the tools in a sample of Android applications are mapped to the vulnerabilities listed by the OWASP project. In the end, it is possible to understand which procedures were adopted for the study, from test execution to tool comparison. The positive and negative aspects of the tools are also noted, which are compared from different perspectives, including precision in detecting vulnerabilities in relation to false positives, time taken to complete the tests, and the ability to detect different types of vulnerabilities from the OWASP Mobile Top 10. Furthermore, the vulnerabilities that were most identified in the context of mobile applications are listed.

Key-words: mobile applications; software security; source code static analysis; vulnerabilities; OWASP.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 Contexto

As aplicações móveis compõem um segmento que cresce constante e rapidamente na área da tecnologia da informação (ISLAM; ISLAM; MAZUMDER, 2010). Isso é evidenciado pela vasta quantidade de áreas para as quais já existem e continuam sendo desenvolvidos produtos de software que agregam algum valor.

Inicialmente, aplicativos móveis, popularmente conhecidos como *apps*, eram desenvolvidos em sua grande maioria para o entretenimento, mas, com o passar dos anos, passaram a atingir outras áreas, como a de sistemas de pagamento, saúde, comunicação, turismo, finanças e educação (MUCCINI; FRANCESCO; ESPOSITO, 2012).

O crescimento apresentado por esse segmento do desenvolvimento de software pode ser justificado pelo fato de que as aplicações móveis têm obtido grande aceitação por parte da sociedade por serem fáceis, acessíveis, possuírem uma usabilidade amigável e poderem ser instaladas na maioria dos dispositivos móveis (ISLAM; ISLAM; MAZUMDER, 2010).

O uso de *apps* está se tornando cada vez mais comum. As pessoas estão habituadas a usá-los para realizar tarefas específicas no seu dia a dia. (ISLAM; ISLAM; MAZUMDER, 2010). São utilizados para executar desde tarefas simples, como criação de lista de compras, organização de compromissos na agenda e compartilhamento de fotos e mensagens de texto com familiares e amigos, até tarefas de natureza mais sensível e crítica como transferências bancárias, compras em lojas virtuais e armazenamento e compartilhamento de documentos importantes. O crescimento da área de desenvolvimento de aplicações móveis também resultou em novas oportunidades no setor público. Diferentes tipos de *apps* foram criados para facilitar o acesso do cliente ao serviço público, além de aplicações criadas principalmente para uso interno em órgãos públicos (GANAPATI, 2015).

Esses *apps* operam com um grande volume de dados sensíveis e a presença de falhas nessas aplicações pode resultar em rastreamento do usuário do serviço, roubos de identidade e perdas financeiras do Estado. Sendo um dos principais meios pelos quais os usuários interagem com a Internet e sistemas de computação, o impacto que causam na segurança da informação é significativo (FAHL, 2021). Constantemente, centenas de criminosos cibernéticos buscam violar as aplicações publicadas em lojas de aplicativos, como Play Store e Apple Store, para conseguir informações do usuário ou embutir nelas algum software malicioso (VELU, 2016).

Segundo Velu (2016), aplicações móveis são, dentre as categorias de aplicações, as que mais possuem falhas de vulnerabilidade, colocando em risco a privacidade e a

segurança de seus usuários. Ataques a estes aplicativos estão associados à maneira com que os usuários os utilizam e com os métodos utilizados pelo *app* para se comunicar com o usuário. Estudos também revelam que o armazenamento e transmissão insegura de dados, injeção do lado do cliente e senhas ou chaves inseridas no código são alguns exemplos de vulnerabilidades que estão relacionadas a essas aplicações (ALANDA et al., 2020).

Diante do crescente uso de aplicações móveis e dos riscos apresentados por essas aplicações por conta de suas vulnerabilidades, faz-se necessário tomar medidas durante o desenvolvimento dessas aplicações, visando proporcionar segurança e confiabilidade para seus usuários.

Uma das práticas frequentemente citadas nos modelos de ciclo de vida de desenvolvimento de software seguro é o teste de segurança estático, bastante conhecido pela sigla SAST, que em inglês significa *Static Application Security Testing*. O SAST consiste em analisar o código fonte das aplicações antes de executá-las em busca de fraquezas, dentre as quais, as vulnerabilidades de segurança (PRAHOFER et al., 2012). Um dos benefícios do SAST é a identificação de falhas na segurança da aplicação ainda nos estágios iniciais de desenvolvimento, permitindo que a correção das falhas seja menos dispendiosa e realizada com um esforço menor em relação a correção de falhas identificadas de maneira tardia (NUNES et al., 2018).

Por prover um meio para identificação de defeitos críticos nas aplicações, o uso da técnica de análise estática de código resulta no aumento da qualidade de software (PRAHOFER et al., 2012). Os benefícios obtidos com o uso dessa técnica podem ser ampliados por meio do uso de ferramentas automatizadas de análise estática de código, proporcionando ganhos como aumento da velocidade, profundidade e precisão da análise.

1.2 Problema

O crescimento do mercado *mobile*, aliado ao fato de que as aplicações são usadas de forma cada vez mais crítica, faz com que a preocupação com sua segurança seja redobrada (MUCCINI; FRANCESCO; ESPOSITO, 2012). As aplicações móveis não estão livres de defeitos ou vulnerabilidades, que podem acontecer por diversos motivos, desde configurações incorretas até erros de implementação no nível de código (VELU, 2016).

Diante disso, é necessário adotar práticas de Engenharia de Software que garantam a segurança e qualidade das aplicações móveis (WASSERMAN, 2010). Para o auxílio dessa atividade, existem algumas ferramentas de software. Dessa forma, esta monografia tem como objetivo responder às seguintes perguntas de pesquisa:

- **Q1**: *Como comparar as ferramentas de teste estático de segurança mais facilmente acessíveis aos desenvolvedores quanto à sua capacidade de detecção de vulnerabili-*

dades em aplicações móveis para Android?

- **Q2**: *Quais são as vulnerabilidades mais comuns presentes em aplicações móveis Android de código aberto?*

1.3 Objetivos

1.3.1 Objetivo Geral

Dado o impacto da segurança das aplicações móveis na privacidade dos seus usuários, o objetivo geral é fazer uma análise comparativa das ferramentas de teste estático de segurança quanto à capacidade de identificação de vulnerabilidades em aplicações Android. Além disso, a partir dos resultados da análise, identificar quais são as vulnerabilidades mais comuns presentes na amostra de aplicações móveis selecionada.

1.3.2 Objetivos Específicos

Este trabalho tem os seguintes objetivos específicos:

- Selecionar ferramentas que podem ser utilizadas por desenvolvedores de software para identificar vulnerabilidades em aplicações móveis para Android;
- Selecionar aplicações móveis de código aberto para o sistema operacional Android;
- Identificar as vulnerabilidades de segurança mais comuns no contexto de aplicações móveis para Android;
- Realizar o teste estático de segurança nas aplicações Android utilizando as ferramentas selecionadas;
- Identificar as diferentes características das ferramentas selecionadas, analisando sua capacidade de detecção de vulnerabilidades em aplicações móveis Android.

1.4 Organização

Este trabalho está organizado nos seguintes capítulos:

- Referencial Teórico: apresenta os principais conceitos do trabalho e a relação de obras e autores que servem de base para a sua elaboração.
- Metodologia: descreve a metodologia a ser utilizada no trabalho para a seleção das aplicações e ferramentas, bem como a análise dos dados.

- Seleção de Ferramentas e Aplicações: detalha as ferramentas de teste estático selecionadas para a comparação e uma visão geral a respeito da utilização de cada uma. Também apresenta as aplicações utilizadas como amostra no estudo.
- Execução dos Testes Estáticos nas Aplicações Selecionadas: descreve o processo de execução das análises, além das configurações e versões utilizadas e resultados iniciais obtidos com a execução.
- Análise de Falsos Positivos: descreve a realização da etapa de análise de falsos positivos e quantidades encontradas.
- Mapeamento das Vulnerabilidades: demonstra o mapeamento das vulnerabilidades por parte das ferramentas, bem como, quando não disponíveis automaticamente, as definições finais resultantes do mapeamento manual.
- Análise dos Resultados: comparação das ferramentas com base nas métricas e dados obtidos por meio da execução das ferramentas de teste estático por meio de gráficos, tabelas e discussões.
- Considerações Finais: apresentação da conclusão a respeito dos resultados obtidos com relação aos objetivos inicialmente definidos e propostas de trabalhos futuros.

2 Referencial Teórico

2.1 Aplicações móveis

As aplicações móveis são arquivos executáveis que podem ser baixados e armazenados no dispositivo do usuário (AUTILI et al., 2021). De uma maneira geral, são softwares executados em componentes eletrônicos móveis e executam tarefas recebendo ou não entradas de informações contextuais (MUCCINI; FRANCESCO; ESPOSITO, 2012).

Há uma grande variedade de aplicações móveis de fácil utilização para atender necessidades específicas, como bate-papo, exames de saúde, jogos, comércio, serviços financeiros, entre outros (VELU, 2016). Os pesquisadores Islam, Islam e Mazumder (2010) listam seis categorias para essas aplicações, de acordo com a área na qual são utilizadas, sendo elas:

- Comunicação - redes sociais e navegadores de Internet.
- Jogos - cartas, cassino, ação e aventura.
- Multimídia - tocadores de vídeo e gravadores de áudio.
- Produtividade - calendário, calculadora e bloco de notas.
- Viagem - conversor de moedas e GPS.
- Utilidades - gerenciador de arquivos e lista de endereços.

As aplicações móveis também podem ser categorizadas em três tipos diferentes: *app* nativo, aquele obtido em lojas de aplicativos; *mobile web app*, páginas da web que são melhoradas para o acesso por meio de dispositivos móveis e *app* híbrido, aquele que conta com a combinação de conteúdo baseado na web e acesso a serviços nativos (VELU, 2016).

Nas subseções seguintes apresenta-se um detalhamento das aplicações nativas e sua instalação.

2.1.1 Aplicação Nativa

Os sistemas operacionais móveis oferecem um conjunto de ferramentas de desenvolvimento de aplicações específicas para o próprio sistema, denominado SDK, do inglês *Software Development Kit*. Por conta disso, essas aplicações são chamadas de nativas do

sistema operacional para o qual foram desenvolvidas (SCHLEIER et al., 2022). No caso do Android, geralmente são escritas na linguagem Java ou Kotlin.

Apps nativos possuem uma estreita interação com o sistema operacional (SCHLEIER et al., 2022), o que lhes confere a capacidade de utilizar recursos tais como câmera, GPS, lista de contatos e armazenamento de chaves com suporte de chaves (VELU, 2016).

Quando comparadas às outras categorias de aplicações móveis, as aplicações nativas apresentam uma maior consistência no que diz respeito à interface do usuário. Isso acontece porque essas aplicações aderem aos princípios de design propostos pelo sistema operacional em que são executadas (SCHLEIER et al., 2022).

2.1.2 Instalação de Aplicativos

Uma das maneiras de instalar *apps* no Android é por meio de um arquivo APK (*Android Application Package*). Esse formato de arquivos instaláveis é utilizado pela Google para distribuir as aplicações para o sistema operacional. São arquivos comprimidos que contém o código relevante da aplicação compilada (VELU, 2016). Ao extrair o conteúdo do arquivo comprimido é possível observar a sua estrutura, que consiste geralmente das seguintes pastas e arquivos:

- assets: diretório com arquivos que devem ser empacotados com o aplicativo
- res: contém recursos necessários para a aplicação que podem ser acessados pelo código de forma estruturada
- lib: contém bibliotecas nativas com recursos que dependem da arquitetura do processador
- META-INF: contém os certificados usados para assinar a aplicação e uma lista dos arquivos incluídos no pacote
- classes.dex: arquivo executável com bytecode Dalvik da aplicação que roda no Dalvik Virtual Machine, responsável por rodar apps programados em Java com restrições limitadas de hardware e memória
- AndroidManifest.xml: arquivo com informações de configuração da aplicação
- resources.arsc: arquivo com recursos pré-compilados, como os arquivos XML da interface dos componentes e *strings* da aplicação

Um dos principais fatores para o sucesso de uma aplicação móvel é o modelo de distribuição adotado por essas lojas, sendo o Google Play Store no caso do Android (AUTILI et al., 2021). O CyBOK as define como plataformas de distribuição digital

centralizada de software em vários ecossistemas web e móveis. Como os aplicativos são distribuídos por provedores de aplicativos, são aplicadas diversas técnicas de verificação de segurança, tanto estática quanto dinâmica, além do processo de assinatura do aplicativo, com o objetivo de certificar a segurança do *app* antes de disponibilizá-lo na loja. Além disso, essas plataformas geralmente disponibilizam funcionalidades para que os usuários dos aplicativos avaliem as aplicações publicadas (FAHL, 2021).

Há também maneiras alternativas de instalar os aplicativos no Android. Uma opção é fazer o download a partir de websites que fazem um espelho dos *apps* que estão na Play Store, porém nesse caso não há garantia de que a aplicação não contém *malware*, e portanto, normalmente devem ser evitados (SCHLEIER et al., 2022).

2.2 Segurança de Software

Segurança de software é uma área ampla e que envolve testar, avaliar, melhorar, impor e prover propriedades de segurança de software (PAYER, 2021). Trata-se de uma área de grande importância e relevância no desenvolvimento de software. Isso se dá por conta do impacto que tem na vida das pessoas, visto que todos os setores da economia global, desde serviços de energia e água a bancos e finanças, transportes e telecomunicações fazem uso ou até dependem de softwares (RANSOME; MISRA, 2013).

É importante destacar a relação entre a segurança de software com a qualidade do produto. A qualidade do código está relacionada ao reuso de código, a facilidade de se fazer manutenção, a correta implementação de funcionalidades, usabilidade e outras questões subjetivas. Por outro lado, a segurança de software não é subjetiva, e está relacionada com a não exposição de dados, controle de acessos e assim por diante. Ainda assim, uma área depende da outra. Esses dois atributos se complementam para aprimorar a integridade e o valor do produto (RANSOME; MISRA, 2013). Por exemplo, um código que impede acessos não autorizados só agregará valor e será utilizado se estiver implementando a funcionalidade corretamente, bem como um código que implementa corretamente uma funcionalidade só apresentará real valor se dados sensíveis não forem expostos.

Para o desenvolvimento de um software seguro, a equipe de desenvolvimento precisa estar preocupada em atingir um objetivo: a segurança da informação. Esse objetivo está relacionado com a proteção das informações e do sistema contra acessos não autorizados, exposições, modificação de dados e assim por diante (RANSOME; MISRA, 2013).

Para que se possa atingir o objetivo de segurança da informação, os princípios básicos de segurança precisam ser atendidos e aplicados durante o ciclo de desenvolvimento de software. Esses princípios são chamados de mecanismos de segurança, conceitos fundamentais de segurança ou atributos de segurança. Esses atributos estão presentes na Figura 1, e são eles: confidencialidade, integridade e disponibilidade (PAYER, 2021).

Figura 1 – Princípios de segurança da informação

Fonte: (VALDIVIA, 2022)

As seções seguintes apresentarão de forma mais detalhada os princípios de segurança.

2.2.1 Confidencialidade

A confidencialidade está relacionada a limitações de acesso. Esse atributo de segurança consiste em garantir que um dado que deve ser protegido não pode ser acessado ou recuperado em um ataque (PAYER, 2021). Por exemplo, esse princípio seria violado em uma situação em que, para uma aplicação de leitura de artigos, possuindo um plano gratuito e outro pago, cujos usuários com plano gratuito tem acesso limitado ao conteúdo dos artigos, um usuário com plano gratuito conseguisse acessar os artigos de forma ilimitada mesmo não tendo esse privilégio dentro do contexto da aplicação.

Os conceitos de autorização e autenticação estão relacionados à confidencialidade da aplicação. Autorização se refere aos privilégios concedidos ao usuário que permitem o acesso aos dados, enquanto autenticação confirma a identidade do usuário (RANSOME; MISRA, 2013).

2.2.2 Integridade

A integridade de um software consiste em limitar as alterações realizadas nos dados da aplicação, impedindo a modificação ou destruição das informações do sistema (RANSOME; MISRA, 2013). Sendo assim, o princípio de integridade diz que aqueles

dados que devem ser protegidos devem permanecer sem modificações em um ataque ao sistema ou por usuários que não têm permissão para fazer essa alteração (PAYER, 2021). Por exemplo, uma aplicação falha quanto a integridade se um usuário sem privilégios de acesso e atualização realiza operações de modificação ou exclusão de dados.

Uma aplicação tem a sua integridade avaliada de acordo com a forma como ela armazena, transmite e aceita dados. Dessa forma, para garantir a integridade, pode-se utilizar técnicas como a criptografia de dados e assinaturas digitais (RANSOME; MISRA, 2013).

2.2.3 Disponibilidade

O princípio de disponibilidade consiste em garantir que o acesso às informações seja obtido sempre que solicitado de forma legítima. Dessa forma, esse princípio será avaliado em questão da porcentagem do tempo em que o software está disponível de acordo com o que foi programado (RANSOME; MISRA, 2013).

A disponibilidade de uma aplicação é quebrada quando o software permite que um ataque impeça o acesso a alguns recursos necessários para a execução da aplicação, tais como memória, rede, processamento ou banco de dados. Por exemplo, esse princípio seria ferido caso um invasor, em seu ataque, consiga desligar o servidor de arquivos, tornando indisponível o acesso aos dados (PAYER, 2021).

2.3 Segurança em Aplicações Móveis

2.3.1 Projetos de Referência

Uma referência na área de segurança de dispositivos móveis é o *Cyber Security Body of Knowledge* (CyBOK), um projeto que visa a codificação de conhecimentos aceitos na área de cibersegurança. Ele é dividido em 21 áreas de conhecimento, sendo uma delas a área denominada "Segurança web e móvel", com foco nas tendências e fenômenos de uso das aplicações móveis e web e nos seus mecanismos de segurança (MARTIN et al., 2021).

Há também o *Open Worldwide Application Security Project* (OWASP), que é uma organização que busca melhorar a segurança do software no aspecto de segurança. Para isso, utiliza projetos *open source* da comunidade, realiza treinamentos educativos, além de disponibilizar outros recursos e ferramentas (OWASP, 2023).

Um dos projetos que fazem parte do OWASP é *Mobile Application Security* (MAS). Alguns dos padrões de segurança disponibilizados pelo MAS são o *Mobile Application Security Verification Standard* (MASVS) e o M*obile Application Security Testing Guide* (MASTG). O MASVS é um framework que fornece diretrizes para monitorar e melhorar a segurança de aplicações móveis, servindo de guia para verificar a segurança dos apps

(SCHLEIER et al., 2023). O MASTG faz um mapeamento do mesmo conjunto de requisitos fornecidos pelo MASVS, e ambos podem ser usados em conjunto para atingir diferentes objetivos. Nesse contexto, o MASVS pode ser usado durante o planejamento e desenho da arquitetura do *app*, enquanto o guia de testes serve para direcionar os testes de segurança manuais durante ou após o desenvolvimento (SCHLEIER et al., 2022).

2.3.2 Controle de Permissões

Nas aplicações Android há a presença de um arquivo chamado AndroidManifest.xml, que se encontra na raiz do APK de um aplicativo, e contém informações sobre a estrutura do *app*, seus componentes e alguns metadados, como o ícone, versão e tema da aplicação, podendo também ter informações sobre APIs compatíveis e tipo de armazenamento onde pode ser instalado (SCHLEIER et al., 2022).

Um dos dados fundamentais para o funcionamento do framework que está presente nesse arquivo xml são as permissões de acesso especial, por meio das quais uma aplicação pode sofrer um ataque que tem como objetivo acessar dados sensíveis do aplicativo. Assim, pode ameaçar a confidencialidade dos dados do usuário (AVANCINI; CECCATO, 2013). Para algumas permissões é utilizado o controle por diálogo, que solicita acesso ao usuário, permitindo proteger a sua privacidade e controlando o acesso dos aplicativos a determinados recursos. No Android há uma distinção entre os dois níveis de acesso, que são baixo (como acesso a internet) e alto (como acesso ao microfone e câmera). Embora algumas dessas permissões sejam solicitadas por diálogo, há também as que são permitidas automaticamente de forma silenciosa (FAHL, 2021).

2.4 Vulnerabilidades

Segundo Shirey (2000), uma vulnerabilidade se trata de uma falha no projeto ou na implementação que pode ser explorada para violar a segurança do sistema. Esse ponto fraco do projeto pode comprometer a confiabilidade, integridade ou disponibilidade da aplicação (ELDER et al., 2022).

Tratando-se de aplicações móveis, Chell et al. (2015) afirmam que as vulnerabilidades mais críticas estão relacionadas à exposição de dados. Isso porque os dados que estão armazenados em um *smartphone* podem ser mais valiosos do que o próprio dispositivo (DWIVEDI; CLARK; THIEL, 2010).

As aplicações móveis possuem uma grande quantidade de vulnerabilidades, sendo algumas delas em comum com as outras categorias de aplicação, como *desktop* ou web. Entretanto, há algumas vulnerabilidades específicas de aplicações móveis oriundas da forma como são utilizadas e dos pontos de ataque únicos apresentados por essas aplicações (CHELL et al., 2015).

Alguns *apps* fazem uso de alguma rede para comunicação. Essa rede, por sua vez, pode não ser segura e confiável. Um exemplo disso é a utilização de redes disponibilizadas por hotéis, shoppings, restaurantes ou outros pontos comerciais que oferecem ao cliente uma rede Wi-Fi como cortesia. Essa utilização apresenta riscos, pois resulta em brechas para exposição de dados sensíveis ou injeção de dados. (CHELL et al., 2015).

As aplicações nativas possuem uma grande variedade de fontes de entradas de informação, como *near field communication* (NFC), *Bluetooth*, câmera e microfone. De uma outra perspectiva, essas diferentes fontes são, na verdade, diferentes pontos de entrada para possíveis ataques às aplicações (CHELL et al., 2015). Além disso, o acesso à Internet expõe o dispositivo a *malwares*, *spywares*, *trojans* e vírus (DWIVEDI; CLARK; THIEL, 2010).

Vírus e *malware* são abundantes no contexto de aplicações móveis. Vez por outra, aplicações oferecem risco ao usuário por tentar obter informações importantes e sigilosas (AU; CHOO, 2016). Isso acontece quando o usuário obtém um aplicativo por meio de mercados de distribuição não oficiais. Sendo assim, o desenvolvedor de aplicativos deve estar ciente de que a sua aplicação pode sofrer ataques de outras aplicações no dispositivo do usuário final (CHELL et al., 2015).

2.4.1 OWASP Mobile Top 10

Um dos projetos mantidos pelo OWASP é o Mobile Top 10. A versão mais recente do Mobile Top 10 disponibilizada pelo OWASP é de 2016, e está presente na Tabela 1. Os riscos listados permanecem os mesmos desde então, sendo que essa última iteração é uma variação dos que foram reportados em 2014 (SCHMEELK; TAO, 2022). Durante o desenvolvimento deste trabalho está sendo elaborada uma nova lista para 2023, porém ainda não foi disponibilizada de forma definitiva.

Tabela 1 – Tabela de riscos do OWASP Top 10

Vulnerabilidade	**Descrição**
M1	Uso Indevido da Plataforma
M2	Armazenamento Inseguro de Dados
M3	Comunicação Insegura
M4	Autenticação Insegura
M5	Criptografia Insuficiente
M6	Autorização Insegura
M7	Qualidade do Código do Cliente
M8	Adulteração de Código
M9	Engenharia Reversa
M10	Funcionalidade Irrelevante

A vulnerabilidade M1, nomeada por uso indevido da plataforma, está relacionada

com a não utilização ou a utilização inapropriada dos recursos disponibilizados pelas plataformas, bem como as falhas nas aplicações em atender aos controles de segurança da plataforma (LEE; VERWER, 2018).

A vulnerabilidade M2, armazenamento inseguro de dados, envolve os defeitos que permitem acesso aos dados armazenados em texto simples por aplicativos, ou ainda em formato alterado utilizando uma chave codificada (CHELL et al., 2015). Os testes para essa vulnerabilidade podem ser feitos pensando no armazenamento interno e externo do dispositivo, arquivos de log, dados XML, dados binários, cookies e banco de dados SQL (QIAN; PARIZI; LO, 2018).

A comunicação insegura, listada como M3 pelo Mobile Top 10, está relacionada com a utilização do serviço de rede por parte da aplicação móvel de forma insegura, ou seja, que permite a interceptação de dados sensíveis por parte de usuários da rede local, dispositivos de redes ou *malware* no próprio dispositivo (BORJA et al., 2021). Essa é identificada como uma das vulnerabilidades mais comuns em aplicações móveis devido a arquitetura cliente-servidor utilizada por essas aplicações. Assim, um *app* pode utilizar um protocolo seguro na comunicação com o servidor para o usuário realizar a autenticação, mas não utilizá-lo em outras partes do software.

A vulnerabilidade de autenticação insegura (M4) está relacionada com o gerenciamento do usuário e ocorre quando há uma falha no método de autenticação utilizado pela aplicação (BORJA et al., 2021). Isso resulta na fragilidade em que um usuário não autenticado pode acessar as informações e funcionalidades do *app* (ALANDA et al., 2020).

A criptografia dos dados de uma aplicação, embora fortemente recomendada, caso seja má implementada não será suficiente para proteger os dados e informações confidenciais. Algumas relações pode ser citadas como as principais para o risco de criptografia insuficiente (M5), sendo elas hardware comprometido, fraquezas e usos inadequados de API, gerenciamento inadequado de chaves e não utilização de criptografia (SCHMEELK; TAO, 2022).

A vulnerabilidade M6, autorização insegura, está relacionada à fraqueza apresentada pelo esquema de autorização da aplicação. Ocorre quando um aplicativo para dispositivo móvel não tem uma verificação dos privilégios do usuário ou essa verificação é fraca (OWASP, 2023). Diante dessa vulnerabilidade, em um ataque, torna-se possível realizar escalonamento de privilégio para utilizar funcionalidades do sistema que exigem um nível de autorização mais alto (BORJA et al., 2021).

A vulnerabilidade de qualidade do código fraca, listada como M7 pelo projeto Mobile Top 10, está relacionada com a forma como o código é escrito, o que faz com que esteja ligada a outras vulnerabilidades (BORJA et al., 2021). Com uma qualidade de código fraca, a aplicação está sujeita a erros evidenciados pelo funcionamento incorreto

das funcionalidades e até pelo sistema parando de funcionar por completo (ALANDA et al., 2020). De acordo com OWASP (2023) essa vulnerabilidade tem um alto esforço para ser detectada e pode ser explorada por invasores por meio de *malware* ou técnicas de *phishing*. Essa vulnerabilidade também está ligada a ataques por exploração de *buffer overflow* e vazamento de memória, permitindo que o atacante execute códigos maliciosos a partir de ataques de injeção de código.

A adulteração de código (M8) consiste na fragilidade da aplicação em permitir que um invasor consiga modificar diretamente o código, alterar o conteúdo da memória dinamicamente ou alterar os dados do *app*, resultando em uma mudança na lógica ou execução da aplicação e permitindo acesso indevido para o benefício pessoal ou monetário do invasor (ALANDA et al., 2020). A exploração dessa vulnerabilidade é comum por meio de ataques de *phishing*. Além disso, um atacante pode explorar essa vulnerabilidade por meio de formulários maliciosos presentes em aplicações hospedadas em lojas de aplicativos de terceiros.

A vulnerabilidade de engenharia reversa (M9) consiste na fragilidade da aplicação perante o uso de ferramentas como descompiladores e depuradores por parte de atacantes para a compreensão do funcionamento da aplicação analisada, objetivando descobrir pontos fracos no código fonte (BORJA et al., 2021).

A funcionalidade irrelevante (M10) não é diretamente apresentada para o usuário final pela interface do *app*, mas pode ser muito útil para um atacante, pois pode fornecer informações com respeito a capacidade do *backend* da aplicação. Uma aplicação móvel pode possuir formas de acesso não documentadas com o objetivo de facilitar as atividades de melhoria e correção do sistema por parte do desenvolvedor (ALANDA et al., 2020). No entanto, um acesso pode ser realizado por esse mesmo meio com intenções maliciosas. Um hacker pode explorar as funcionalidades escondidas no *backend* da aplicação para realizar um ataque.

2.4.2 CWE

A *Common Weakness Enumeration*[1] (CWE) consiste em uma lista de tipos de fraquezas de software e hardware desenvolvida pela comunidade. Trata-se de um esforço conjunto entre o Departamento de Segurança Interna dos Estados Unidos, a Agência de Segurança Nacional dos Estados Unidos e a comunidade de software, sendo que a a Corporação MITRE é líder técnica e coordena o projeto, que possui um catálogo de mais de 800 fraquezas de software (RANSOME; MISRA, 2013).

Uma importante vantagem do CWE é que diversas ferramentas de segurança e de detecção de vulnerabilidades são capazes de apresentar o CWE relacionado aos tipos das

[1] https://cwe.mitre.org/

vulnerabilidades identificadas (ELDER et al., 2022).

Alguns exemplos de CWE que podem representar riscos para as aplicações mobile são o de exportação incorreta de componentes de aplicativos Android (CWE 926), socket não criptografado (CWE 319) e uso de *intent* implícita para comunicação sensível (CWE 927) (OYETOYAN; CHAIM, 2017).

2.5 Análise Estática de Código

A análise de um código pode ser feita de diferentes abordagens, como a estática e a dinâmica, a fim de encontrar problemas de segurança em um software. A análise estática recebe como entrada o código fonte ou binário de um programa e o analisa sem precisar executá-lo, sendo a eficiência e a velocidade os principais motivos pelos quais essa técnica é amplamente adotada (PAN et al., 2020). A análise dinâmica, por sua vez, é utilizada para monitorar a aplicação enquanto está sendo executada. Sendo assim, ela se diferencia da dinâmica por poder ser realizada sem a necessidade de o software estar em execução.

A análise estática possui vários benefícios para a segurança de um software, uma vez que, sem precisar executar a aplicação, ela encontra vulnerabilidades examinando todas as possíveis rotas e valores de variáveis (BASUTAKARA, 2021). Além disso, principalmente quando comparada com a análise dinâmica, possui alta eficiência de execução e alta velocidade (PAN et al., 2020). Pode-se citar também que, para aplicá-la, não há necessidade de alterar o código da aplicação e que as ferramentas que a utilizam podem ser facilmente implantadas como aplicativos padrão no dispositivo (KULKARNI; JAVAID, 2018).

Apesar da análise estática ser capaz de fornecer importantes informações a respeito de um software, ela também possui limitações. Embora a análise estática seja rápida, ela pode não ser eficaz na detecção de malwares criptografados, polimórficos e com código alterado (KULKARNI; JAVAID, 2018). Além disso, caso muitos recursos sejam empregados na análise, a detecção de falsos positivos e falsos negativos pode ser inevitável.

No caso do Android, por ser um sistema operacional de código aberto, há algumas importantes vantagens ao realizar a análise estática. Ele oferece maior controle e flexibilidade, alimentando um ecossistema de ferramentas e bibliotecas úteis para análise estática, além de ser a principal plataforma para abordagens genéricas de análise estática (AUTILI et al., 2021). De qualquer maneira é essencial configurar as ferramentas de forma apropriada e selecionar categorias específicas para a análise, pois os resultados gerados por elas podem ser excessivos e contraprodutivos para análise manual (SCHLEIER et al., 2022).

3 Metodologia

Esse capítulo tem como objetivo detalhar a metodologia utilizada para realizar uma análise comparativa de ferramentas de teste estático em aplicações móveis para a plataforma Android e identificar as suas vulnerabilidades mais comuns. A pesquisa realizada foi classificada quanto à natureza, ao objetivo, à abordagem e aos procedimentos.

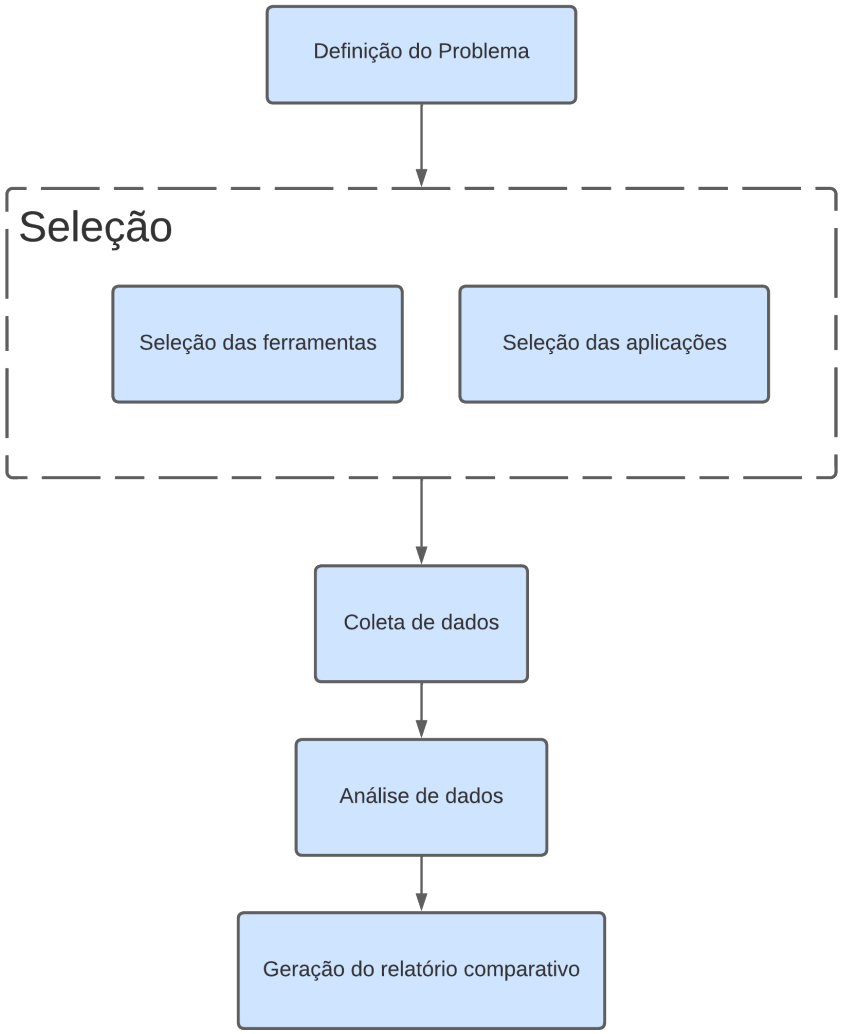
Quanto à natureza, a pesquisa pode ser classificada como aplicada, pois se pretende gerar novos conhecimentos de aplicação prática para a área de segurança em aplicações Android, fornecendo assim informações úteis aos desenvolvedores e auxiliando na tomada de decisões com relação ao uso de ferramentas de análise estática dentro desse contexto.

Quanto ao objetivo, pode ser classificada como descritiva, pois a comparação fornece uma visão sobre as ferramentas de análise estática e suas capacidades na detecção de vulnerabilidades em aplicações Android a partir dos dados obtidos com a execução da análise estática, a fim de aprimorar a segurança dessas aplicações.

Quanto à abordagem, pode ser classificada como quantitativa e qualitativa, pois as métricas quantitativas obtidas com o uso das ferramentas, incluindo vulnerabilidades detectadas e tempo de execução, guiarão a comparação entre elas, permitindo a avaliação da eficácia de cada uma. Assim, pretende-se obter uma visão imparcial e precisa do desempenho das ferramentas. É considerada também qualitativa porque envolve a análise da facilidade de uso das ferramentas a partir dos relatos a serem gerados pelos próprios autores.

Quanto aos procedimentos, enquadram-se a revisão bibliográfica e o estudo de múltiplos casos. A revisão bibliográfica foi realizada para uma melhor compreensão dos conceitos relacionados ao sistema operacional para o qual os apps são desenvolvidos e a respeito de fundamentos de segurança de software. O estudo de múltiplos casos é adotado na comparação das ferramentas selecionadas dentro do cenário das aplicações Android planejadas.

Figura 2 – Fluxograma da metodologia



Fonte: Elaboração Própria

O plano metodológico foi baseado no protocolo para estudo de caso proposto por Brereton et al. (2008), adaptando-o às necessidades deste estudo, conforme apresentado na Figura 2. Para apresentar de forma detalhada a metodologia utilizada, esse capítulo está organizado de acordo com as seguintes seções:

- Definição do problema
- Seleção das ferramentas e aplicações
- Coleta de dados
- Análise de dados
- Limitações do estudo
- Relatório

- Cronograma

A Tabela 2 apresenta o mapeamento das seções propostas pelo protocolo utilizado como base para as seções apresentadas nesta metodologia.

Tabela 2 – Mapeamento das seções

Seção (protocolo base)	**Seção (metodologia)**
Background	Definição do problema
Design	Definição do problema
Selection	Seleção das ferramentas e aplicações
Procedures and roles	Seleção das ferramentas e aplicações
Data collection	Coleta de dados
Analysis	Análise de dados
Plan validity	Análise de dados
Study limitations	Limitações do estudo
Reporting	Relatório
Schedule	Não se aplica
Appendices	Apêndices

3.1 Definição do Problema

Essa etapa, realizada por meio do procedimento de pesquisa bibliográfica, tem como objetivo identificar as lacunas para a elaboração da pergunta de pesquisa, definir os objetivos da pesquisa e apresentar considerações importantes relacionadas à segurança em aplicações móveis, contextualizando a questão de pesquisa.

Com base no procedimento de pesquisa bibliográfica, são apresentadas as principais vulnerabilidades existentes no contexto de aplicações móveis, as quais são listadas pelo programa Mobile Top 10 mantido pela OWASP. Além disso, apresenta-se o impacto dessas vulnerabilidades na segurança da aplicação e, consequentemente, na sua qualidade.

Diante do problema apresentado, observa-se que a atividade de análise estática de código é útil para os desenvolvedores, visto que tem como objetivo identificar as vulnerabilidades presentes no código fonte de uma aplicação. Dessa forma, o uso de ferramentas de análise estática é capaz de auxiliar o desenvolvimento de aplicações, visando diminuir os riscos apresentados.

Este estudo propõe um comparativo entre as ferramentas de análise estática de código por meio da análise dos resultados da execução dessas ferramentas em uma amostra de aplicações Android, o que permite a identificação das ferramentas mais eficazes na detecção das vulnerabilidades de segurança.

3.2 Seleção das Ferramentas e Aplicações

Nessa etapa, realiza-se a tomada de decisões com respeito a quantidade de aplicações Android a serem selecionadas como amostra para a pesquisa e a quantidade de ferramentas de análise estática que serão utilizadas para a identificação das vulnerabilidades presentes nas aplicações da amostra. As quantidades definidas são as seguintes:

- quantidade de ferramentas de análise estática: quatro;
- quantidade de aplicações Android: cinco.

Assim, é esperado que seja obtida uma quantidade adequada de dados para serem analisados, uma vez que cada ferramenta será executada para gerar um relatório para cada aplicação selecionada. Além disso, é exigida uma quantidade de trabalho manual para cada relatório obtido pelas análises das ferramentas.

Outra atividade importante nessa etapa do processo metodológico é a definição dos critérios utilizados na seleção das ferramentas e das aplicações, bem como a atividade de escolha delas.

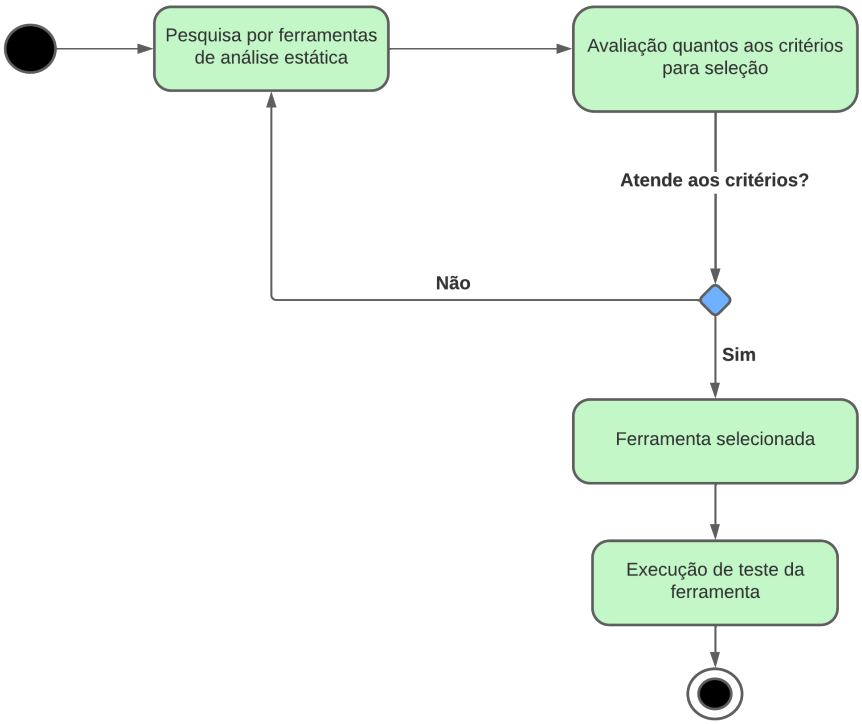
3.2.1 Ferramentas

Para a realização da atividade de seleção das ferramentas de análise estática, foram definidos os seguintes critérios:

- deve possuir alguma atualização no período de seis meses, indicando que se trata de um projeto ativo;
- deve ser gratuita, mesmo que seja apenas uma versão;
- ser capaz de analisar código Java ou Kotlin e
- ser capaz de detectar vulnerabilidades relacionadas à segurança.

A Figura 3 apresenta o método utilizado para a seleção das ferramentas. Primeiramente, foi feita uma busca de ferramentas gratuitas de análise estática de código para aplicações móveis por meio de artigos, blogs e outras fontes na internet. Após isso, observou-se as últimas atualizações feitas na ferramenta e se essa possuía, dentre as funcionalidades da ferramenta, a capacidade de se analisar códigos Java ou Kotlin. Por fim, realizou-se uma execução da ferramenta para testá-la.

Figura 3 – Método de seleção das ferramentas



Fonte: Elaboração Própria

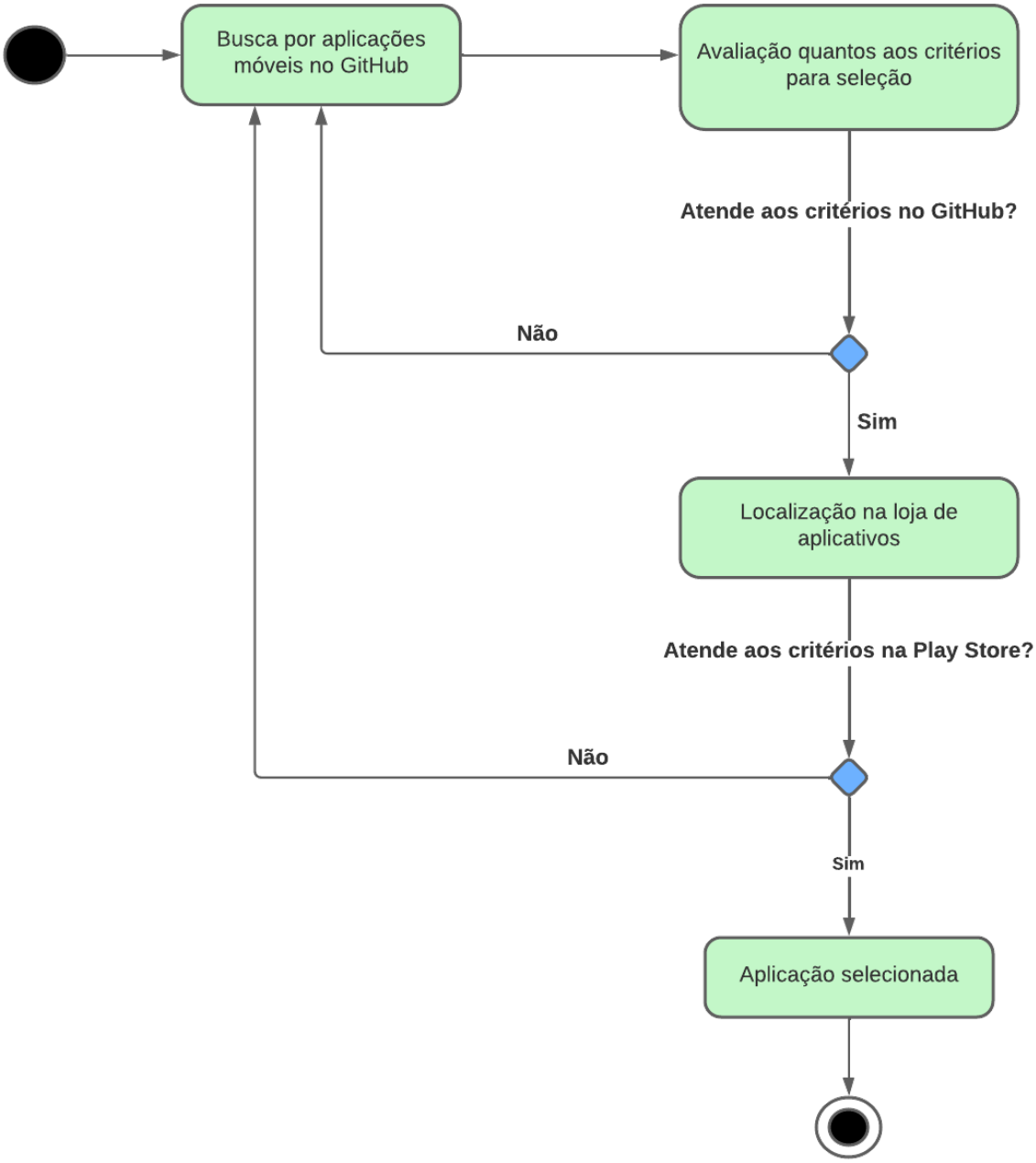
3.2.2 Aplicações

Quanto às aplicações, os seguintes critérios foram estabelecidos para a realização da atividade de seleção:

- deve ser *software* livre;
- deve ser uma aplicação nativa;
- o código fonte da aplicação deve estar disponível no GitHub;
- deve possuir um número de downloads superior ou igual a cem mil, sendo assim considerada relevante;
- deve ser de uma categoria diferente das demais aplicações selecionadas até o momento e
- o número de contribuidores do projeto deve ser superior a 30.

A Figura 4 mostra o método que será seguido para a escolha das aplicações.

Figura 4 – Método de seleção das aplicações



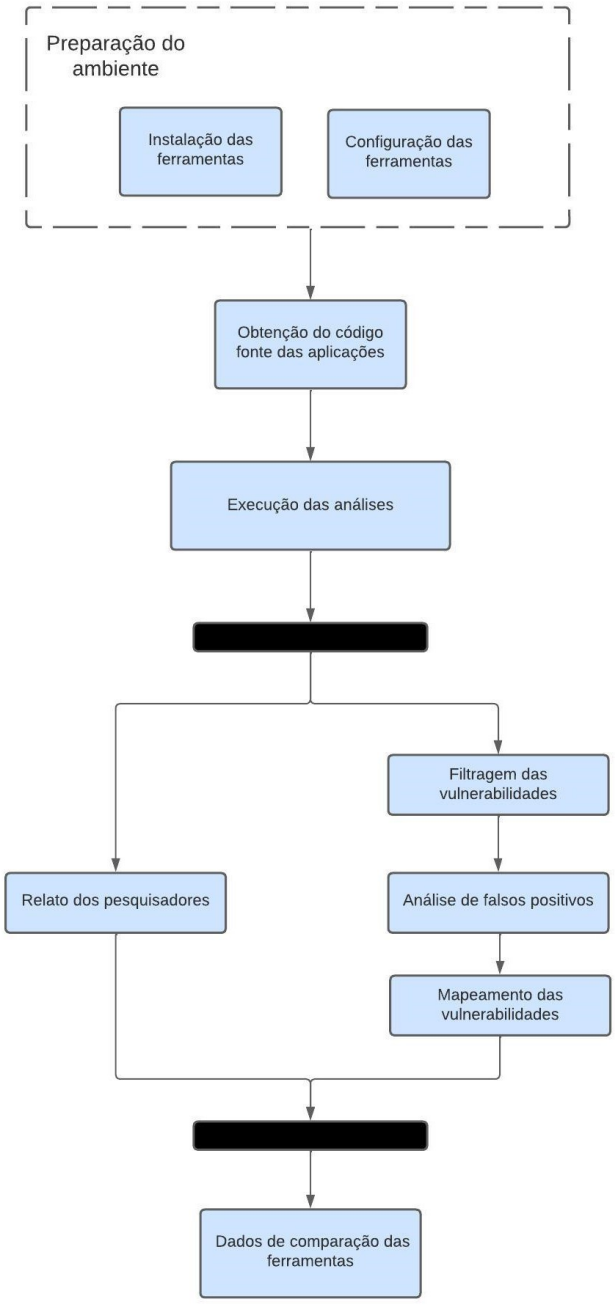
Fonte: Elaboração Própria

As aplicações foram selecionadas a partir de uma busca por aplicações móveis nos repositórios do GitHub. O segundo passo consistiu em analisar o número de contribuidores e se é nativa para Android. Por fim, analisou-se o número de downloads da aplicação realizados na loja de aplicativos disponíveis na Play Store e a categoria à qual a aplicação pertence.

3.3 Coleta de Dados

Essa etapa consistiu em obter as vulnerabilidades das aplicações selecionadas para o estudo. Para isso, utilizou-se o método de análise documental, tendo como documentos de análise os relatórios gerados pelas ferramentas de análise estática de código que foram selecionadas. Essa seção descreve como foi realizada a coleta dos dados e as etapas envolvidas, as quais podem ser observadas na Figura 5 .

Figura 5 – Processo de coleta de dados



Fonte: Elaboração Própria

Inicialmente foi feita a instalação e configuração das ferramentas na máquina na qual as análises seriam executadas. Depois disso, foi obtido pelo GitHub o código fonte de cada aplicação selecionada a partir de seu repositório oficial. Para as ferramentas que são executadas na nuvem, foi feito um *fork* do repositório original da aplicação, no qual se adicionou uma *pipeline* com as configurações da ferramenta.

Com essas etapas concluídas, foi realizada a análise estática com as ferramentas previamente definidas, utilizando como entrada o código fonte das aplicações, em busca de vulnerabilidades de segurança. Assim, como saída dessas análises, foram obtidos os relatórios, gerados pelas próprias ferramentas de análise estática, que guiaram a análise para a comparação.

Os dados de interesse obtidos pelo relatório foram: o tempo despendido pela ferramenta para realizar a análise estática, o número de vulnerabilidades detectadas, o grau de severidade das vulnerabilidades encontradas e as categorias a elas relacionadas.

Outro dado considerado foi a experiência dos pesquisadores em relação ao esforço para a configuração da ferramenta para que seja possível avaliá-la quanto ao grau de dificuldade de preparação do ambiente de análise e obtenção dos resultados. Esse dado foi obtido por meio de relatos dos pesquisadores sobre as dificuldades encontradas quanto à configuração, uso da ferramenta, funcionalidades disponibilizadas e entendimento dos relatórios de análise obtidos. Por fim, cada pesquisador forneceu uma nota para a ferramenta de 1 a 5.

Dentro do contexto deste estudo, é importante levar em consideração que nem todas as ferramentas testadas tem foco em vulnerabilidades de segurança. Logo, no caso dessas ferramentas, houve a necessidade de filtrar as vulnerabilidades obtidas para aquelas que eram de segurança. Em alguns casos isso pode ser configurado na ferramenta antes da execução da análise.

Após a identificação das vulnerabilidades encontradas que estão relacionadas a característica de segurança, cada um dos autores fez uma análise independente em busca de possíveis falsos positivos, pois esses casos devem ser separados na comparação das ferramentas com relação ao aspecto de capacidade de detecção de vulnerabilidades.

Para as ferramentas que já indicam a vulnerabilidade do OWASP Mobile Top 10 não houve necessidade de etapas adicionais, porém para aquelas que não fornecem essa funcionalidade, analisou-se a possibilidade de mapeá-las. Para isso, cada vulnerabilidade do OWASP Mobile Top 10 foi mapeada para um CWE, bem como cada vulnerabilidade encontrada pelas ferramentas. Esse mapeamento foi feito por meio de buscas no catálogo oficial do CWE por vulnerabilidades que pudessem ser relacionadas.

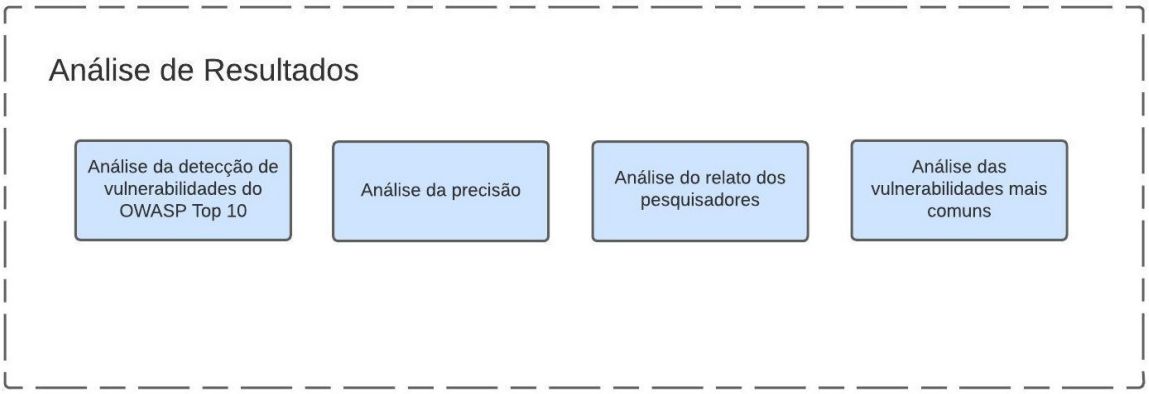
Com o objetivo de prover uma maior confiabilidade da análise, após a identificação individual por parte dos autores dos falsos positivos, as classificações inconsistentes foram

analisadas pelos dois autores e alinhadas para que se obtivesse uma única lista de falsos positivos.

3.4 Análise de Resultados

Nesta seção é detalhado o processo de análise dos resultados conforme apresentado na Figura 6. Primeiramente, é explicado o processo de análise dos resultados obtidos para a comparação das ferramentas com relação a sua capacidade de detecção de vulnerabilidades do OWASP Mobile Top 10. Em seguida, é descrito o processo de identificação das vulnerabilidades mais comuns detectadas com a execução das ferramentas.

Figura 6 – Diagrama de análise dos resultados



Fonte: Elaboração Própria

Para a comparação das ferramentas, os dados foram organizados em tabelas e gráficos, fornecendo uma melhor visualização dos dados. Os dados de cada tabela consistem na quantidade de vulnerabilidades encontradas por cada ferramenta para essa aplicação dentre aquelas disponíveis no OWASP Mobile Top 10. Foi comparada também por meio de uma tabela a quantidade de falsos positivos detectados por cada ferramenta, a partir de análise realizada anteriormente.

Foram classificadas também as ferramentas de acordo com os relatos fornecidos pelos pesquisadores, que são os autores deste trabalho. A partir dos registros sobre as percepções em relação à facilidade de configuração e obtenção de resultados, pretende-se chegar a uma conclusão de quais são as ferramentas mais fáceis de usar de acordo com a experiência na pesquisa, permitindo uma conclusão embasada sobre as ferramentas mais acessíveis.

A execução desses procedimentos de forma padronizada objetivou a obtenção de dados confiáveis e válidos, permitindo uma comparação precisa entre as ferramentas.

Quanto à identificação das vulnerabilidades mais comuns, foram utilizados gráficos para apresentar o índice de ocorrência das diferentes categorias de vulnerabilidades apresentadas pelo Mobile Top 10 nas aplicações móveis selecionadas.

3.5 Limitações do Estudo

O estudo se limitou a uma quantidade reduzida de aplicações, pois se tornaria inviável a análise dos resultados para uma quantidade muito grande, visto que as atividades de configuração das ferramentas para cada aplicação e análise de falsos positivos tendem a exigir muito tempo para serem realizadas. Como tentativa de contornar esse problema, as aplicações foram selecionadas de modo a garantir a presença de categorias variadas, a fim de abordar uma maior variedade de tipos de aplicações.

Pode ser citado como limitação também o fato de que as aplicações selecionadas são restritas às aplicações nativas de Android. Sendo assim, os resultados não necessariamente refletem o cenário das aplicações móveis web e híbridas.

Outra limitação identificada é a utilização da listagem de 2016 das vulnerabilidades mais comuns feita pela OWASP no programa Mobile Top 10. Dessa forma, a ferramenta de análise estática apresentada como a que mais detecta vulnerabilidades para a listagem de referência pode não ser a mesma para novas versões dessa listagem, que infelizmente ainda não estão disponíveis em versão definitiva.

3.6 Relatório

Essa etapa consiste em detalhar os resultados da pesquisa realizada, apresentando o comparativo da capacidade de identificação de vulnerabilidades e facilidade de uso das ferramentas de análise estática selecionadas. Dessa forma, buscou-se apresentar uma análise das ferramentas selecionadas de diferentes perspectivas. Além disso, foram identificadas as vulnerabilidades mais comuns dentro da amostra das aplicações selecionadas para o estudo.

3.7 Apêndices

Essa seção contém exemplos de relatórios gerados e exportados nos formatos suportados pelas ferramentas, além de capturas de tela do processo. Há também a disponibilização das planilhas utilizadas para auxiliar na filtragem de falsos positivos e mapeamento das vulnerabilidades.

4 Seleção de Ferramentas e Aplicações

4.1 Ferramentas Selecionadas

Com base nos critérios estabelecidos, quatro ferramentas de teste estático foram selecionadas. Além disso, realizou-se uma configuração inicial das ferramentas com o objetivo de se obter um relato inicial por parte dos pesquisadores sobre a experiência de uso das ferramentas quanto ao nível de esforço para sua configuração.

As seções seguintes introduzem as ferramentas selecionadas, apresentando uma visão geral sobre elas, os resultados que podem ser obtidos a partir de sua execução e o relato dos pesquisadores sobre a sua utilização.

4.1.1 SonarCloud

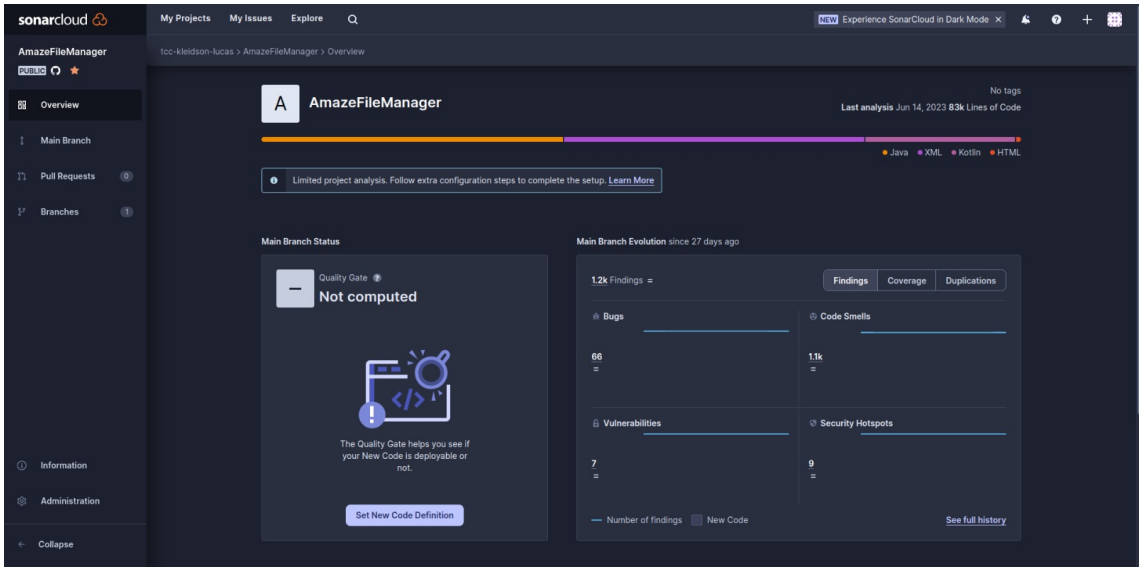
O SonarCloud[1] é uma ferramenta de análise de código baseada em nuvem que permite detectar problemas no código em uma grande variedade de linguagens de programação, incluindo Java e Kotlin. Assim, permite analisar características do código, como manutenibilidade, confiabilidade e segurança.

A ferramenta conta com um *Dashboard* que apresenta os projetos que estão configurados para o uso da ferramenta. Ao executar a ferramenta na aplicação Amaze File Manager[2], obteve-se o relatório apresentado na Figura 7.

O relatório apresenta um quadro com os dados do que foi encontrado na análise estática, apresentando os seguintes tópicos: número de *code smells*, ou seja, fraquezas de arquitetura que podem retardar o desenvolvimento ou aumentar o risco de *bugs* ou falhas no futuro; *bugs*; vulnerabilidades e trechos de código sensíveis à segurança encontrados. Além disso, é possível encontrar mais detalhes sobre cada um desses pontos ao acessar o link presente no indicador de quantidade de cada tópico, conforme ilustrado na Figura 8. Isso permite a identificação da falha e a sua severidade.

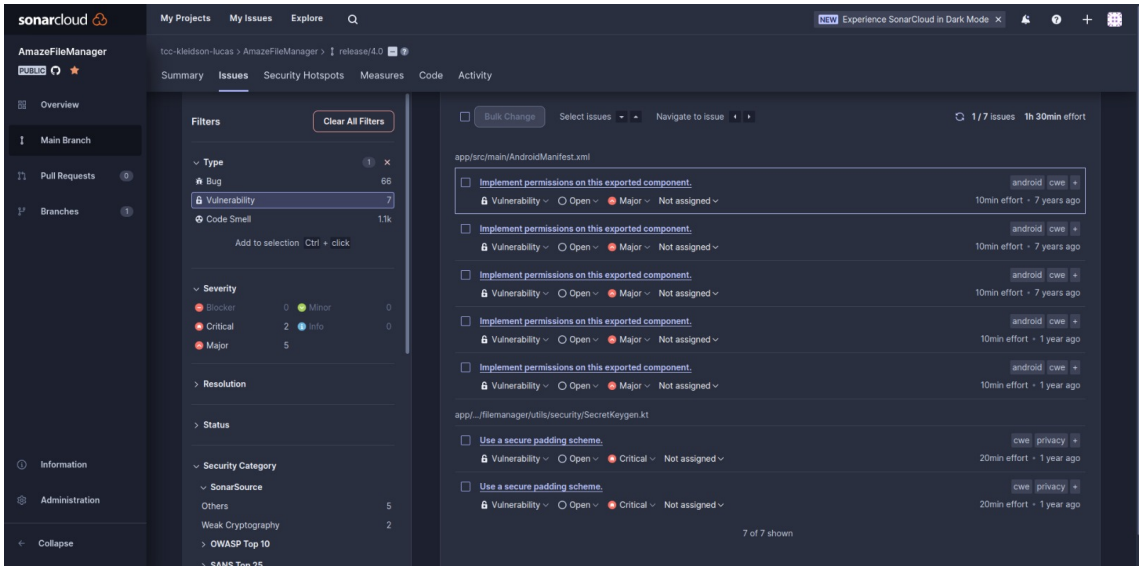
[1] https://www.sonarsource.com/products/sonarcloud
[2] repositório encontrado em: https://github.com/TeamAmaze/AmazeFileManager

Figura 7 – Relatório gerado pelo SonarCloud para a aplicação Amaze File Manager



Fonte: Elaboração Própria

Figura 8 – Vulnerabilidades encontradas na aplicação Amaze File Manager



Fonte: Elaboração Própria

4.1.1.1 Relato dos Pesquisadores

Dentre as ferramentas selecionadas, o SonarCloud foi a que exigiu mais esforço para realizar a configuração. Devido a sua execução ser realizada em nuvem, foi necessário criar uma *pipeline* para realizar a análise estática a cada atualização feita no repositório. No entanto, também é possível executar a análise manualmente.

O SonarCloud possui uma interface amigável e de fácil utilização. A ferramenta conta com uma análise automática que é útil para os desenvolvedores, porém ela é limitada. Também há instruções claras sobre como configurar o projeto utilizando tecnologias

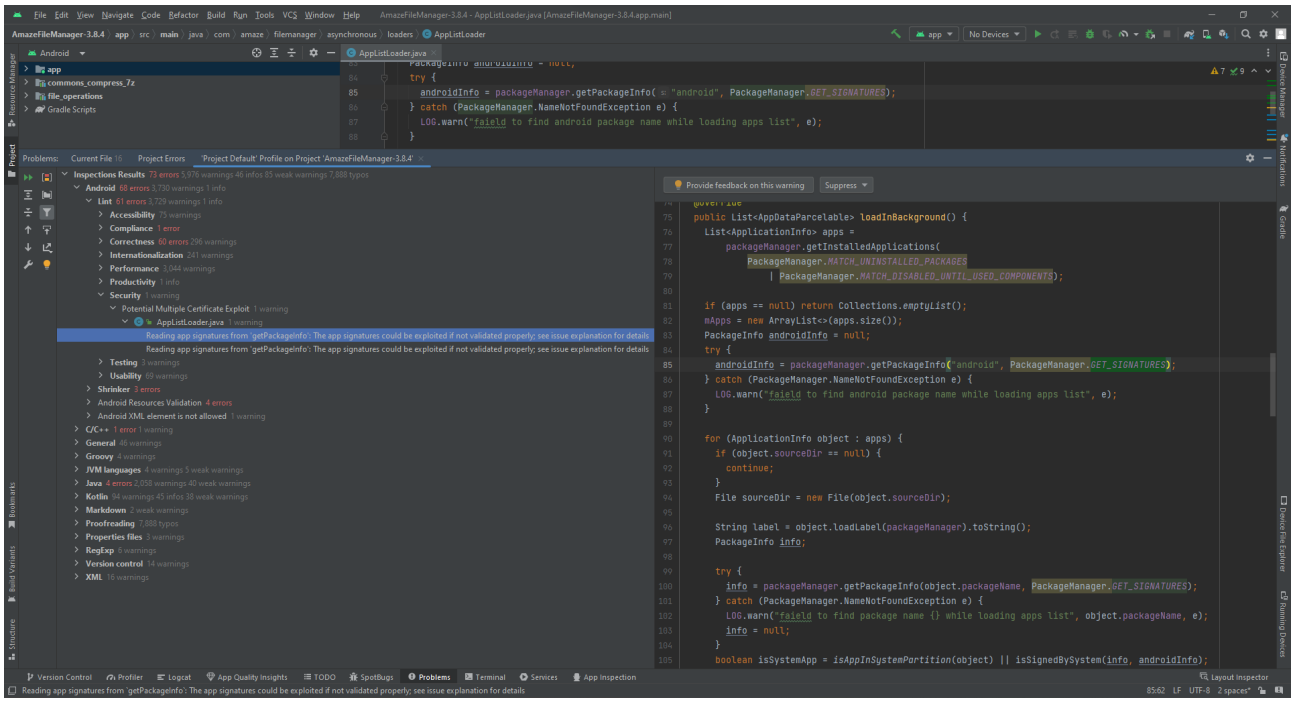
como Maven, Gradle, entre outras. No entanto, para alguns projetos é necessário realizar alguns ajustes adicionais em relação a essas instruções para que funcione corretamente.

4.1.2 Android Lint

A ferramenta de análise estática Android Lint[3] é oferecida pela IDE Android Studio e auxilia na identificação de problemas de qualidade do código. É capaz de encontrar *bugs* ou melhorias relacionadas a diversos critérios, incluindo de segurança. A ferramenta fornece também outras informações importantes, como o nível de gravidade do problema encontrado. Além disso, permite atualizar o nível de gravidade para destacar diferentes tipos de problema.

Na Figura 9, observa-se as diversas categorias e vulnerabilidades encontradas pela ferramenta Android Lint, após execução da análise no IDE Android Studio, para o qual a ferramenta já vem integrada.

Figura 9 – Vulnerabilidades encontradas pela ferramenta Android Lint



Fonte: Elaboração Própria

4.1.2.1 Relato dos Pesquisadores

Por já ser integrada ao Android Studio, a ferramenta Android Lint é bem simples de usar. É necessário apenas fazer a instalação do IDE e, com o projeto aberto, abrir o menu de contexto no módulo que se deseja realizar a análise e selecionar a opção Inspect Code. Após finalizar a análise, será mostrada a lista com as categorias, subcategorias e vulnerabilidades identificadas.

[3] https://developer.android.com/studio/write/lint?hl=pt-br

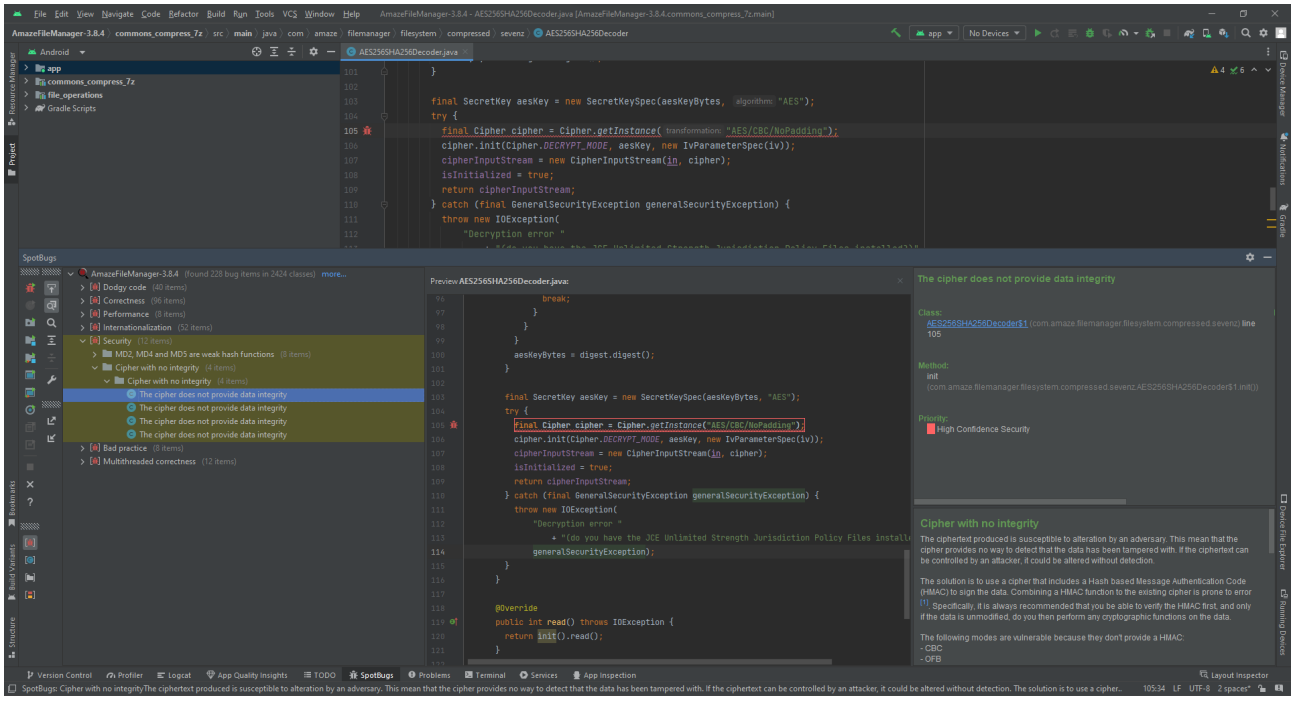
Essa ferramenta possui diversas opções de customização, incluindo a filtragem por categorias e subcategorias para as vulnerabilidades. Além disso, há a possibilidade de agrupá-las de diferentes maneiras após a análise, como por diretório ou por severidade.

4.1.3 FindSecBugs

O FindSecBugs[4] é um plugin para o SpotBugs, um programa de análise estática utilizado para encontrar *bugs* em código Java e que dá continuidade ao antigo FindBugs, que teve o seu suporte encerrado. É capaz de encontrar 141 diferentes tipos de vulnerabilidades e oferece suporte a integração em IDEs, além de referenciar vulnerabilidades do OWASP Top 10 e CWE.

Uma das maneiras de utilizar a ferramenta é por meio da instalação do plugin do SpotBugs disponível no Marketplace do Android Studio. Esse plugin já inclui a extensão FindSecBugs, que adiciona uma seção de detecção de vulnerabilidades de segurança. Na Figura 10 é possível visualizar as vulnerabilidades agrupadas por categoria na seção SpotBugs. Para as vulnerabilidades encontradas a ferramenta fornece a sua descrição, incluindo também informações como a solução para o problema, o CWE associado, entre outras referências que detalham melhor a vulnerabilidade.

Figura 10 – Vulnerabilidades encontradas pela ferramenta FindSecBugs



Fonte: Elaboração Própria

[4] https://find-sec-bugs.github.io/

4.1.3.1 Relato dos Pesquisadores

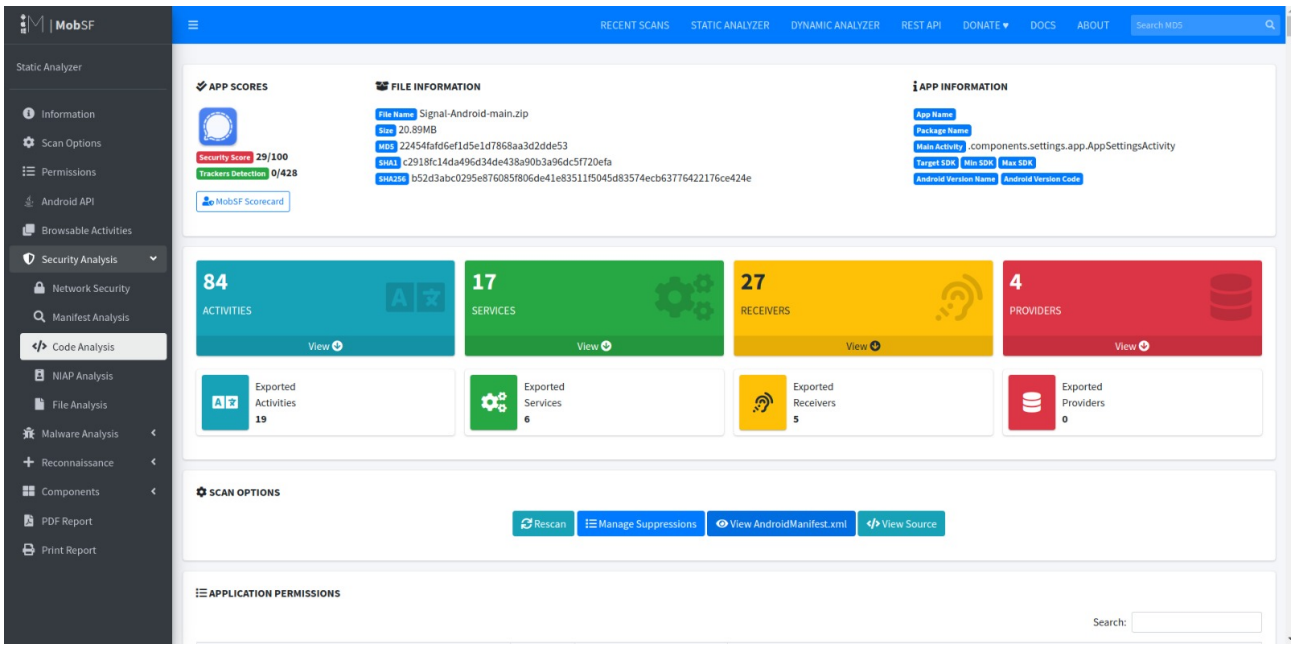
Com a instalação do Android Studio e do plugin SpotBugs, basta abrir o projeto da aplicação e habilitar a extensão FindSecBugs nas configurações de ferramenta do IDE. Depois disso é possível executar a análise a partir do módulo desejado do projeto importado, abrindo o menu de contexto desse módulo e iniciando a análise na opção SpotBugs. Assim, é mostrada a lista com as vulnerabilidades identificadas, que podem variar de acordo com o nível de sensibilidade selecionado nas configurações da ferramenta. Além disso, é possível filtrar apenas as categorias desejadas antes de realizar a análise, o que agiliza o processo de análise feito pela ferramenta.

Por não ser uma opção nativa do Android Studio, pode ser um pouco mais complexo de instalar e configurar, porém o fato de poder ser integrada a IDE assim como o Android Lint faz com que a ferramenta seja de fácil uso.

4.1.4 MobSF

O Mobile Security Framework[5] (MobSF) é uma ferramenta capaz de executar análise estática e dinâmica com foco em aplicações móveis. Pode realizar testes de penetração, análise de *malware* e assessoria de segurança, podendo ser integrado ao processo de CI/CD e DevSecOps.

Figura 11 – Relatório gerado pelo MobSF



Fonte: Elaboração Própria

A Figura 11 apresenta um exemplo do relatório gerado pelo MobSF, no qual é possível observar alguns dados importantes para o trabalho. Inicialmente, o relatório apre-

[5] https://mobsf.github.io/docs

senta uma pontuação da própria ferramenta para a aplicação analisada, bem como as informações a respeito dela. O relatório está dividido em seções, dentre as quais, as seções de análise de segurança e análise de *malware*.

4.1.4.1 Relato dos Pesquisadores

Após a instalação da ferramenta, o seu uso é bem simples. É necessário apenas arrastar o arquivo ZIP do código da aplicação para a janela do navegador onde está sendo executada a ferramenta e esperar a análise ser realizada.

O relatório gerado pela aplicação é bem completo e tende a ser extenso, pois depende do tamanho da aplicação analisada. Ele está bem organizado em diferentes seções, o que facilita o desenvolvedor a se localizar ao analisar os dados apresentados no relatório.

4.2 Aplicações Selecionadas

Foram selecionadas diversas aplicações, a partir da ferramenta de busca do GitHub, dentre aquelas que se encontram no tópico Android filtradas pelas linguagens Java e Kotlin, correspondendo assim às aplicações nativas de Android. Os demais critérios definidos foram analisados manualmente por meio das informações disponíveis no repositório da aplicação no GitHub e na página da aplicação na Play Store. Em seguida essas aplicações foram classificadas por suas categorias com base na categoria informada na loja de aplicativos, a fim de atingir o critério referente à variedade de categorias das aplicações. Por fim, foram selecionadas as cinco aplicações móveis, cada uma pertencendo a uma categoria diferente das demais.

4.2.1 Aegis Authenticator - 2FA App

O Aegis Authenticator[6] é uma aplicação de código aberto para gerenciar *tokens* de autenticação em duas etapas, de maneira similar a outras aplicações populares, como o Google Authenticator e o Microsoft Authenticator. Ela suporta os principais algoritmos da indústria e possui criptografia, além de oferecer diversas funcionalidades, como desbloqueio por digital, digitalização de códigos QR e importação de dados de outros aplicativos autenticadores. O aplicativo, desenvolvido em Java, possui mais de 100 mil downloads na Play Store. O Aegis Authenticator se enquadra na categoria de segurança.

4.2.2 Amaze File Manager

O Amaze File Manager[7] é uma poderosa ferramenta de gerenciamento de arquivos, contando com operações básicas, como explorar diretórios, mover pastas e arquivos,

[6] https://play.google.com/store/apps/details?id=com.beemdevelopment.aegis
[7] https://play.google.com/store/apps/details?id=com.amaze.filemanager

renomear documentos, copiar, colar, deletar, dentre outras operações. A aplicação possui mais de um milhão de downloads na Play Store e se enquadra na categoria de utilidades. Além disso, é desenvolvida em Java e Kotlin e seu repositório do GitHub conta com 133 contribuidores.

4.2.3 Retro Music Player

O Retro Music Player[8] é uma aplicação para reprodução de músicas armazenadas no dispositivo Android, enquadrando-se então na categoria de aplicações de mídia. É desenvolvido em Kotlin e foi projetado para ser simples e fácil de usar. Possui opções de customização, como variedade de temas e paleta de cores, além de recursos como widgets, temporizador, filtro de duração e reprodução sem intervalos. O aplicativo foi baixado mais de um milhão de vezes na Play Store.

4.2.4 Omni Notes

O Omni Notes[9] é uma aplicação desenvolvida em Java para realizar e gerenciar anotações com a proposta de ser simples e de ter boa performance. Suas funcionalidades incluem uma lixeira integrada, adição, remoção e edição de notas, inclusão de anexos nas notas, além de importação e exportação de anotações. O download da aplicação já foi feito mais de 100 mil vezes na Play Store, que o classifica como uma aplicação voltada para a categoria de produtividade.

4.2.5 Wikipédia

A Wikipédia[10] é uma aplicação desenvolvida em Kotlin e tem como objetivo servir como referência para o desenvolvimento de trabalhos. Trata-se de uma enciclopédia livre que contém mais de 32 milhões de artigos em 280 idiomas. O seu projeto no GitHub conta com mais de 100 colaboradores. A aplicação possui mais de 50 milhões de downloads na Play Store e está categorizada como uma aplicação de livros e referências.

4.2.6 Características das Aplicações

A tabela 3 apresenta um resumo das principais características de cada aplicação da amostra selecionada. Incluem-se as informações já mencionadas, além de alguns detalhes adicionais. Os dados incluem a nota da aplicação (entre 1 e 5), a quantidade de avaliações, a categoria da aplicação e o número de downloads, obtidos da loja de aplicativos Play Store. Também são apresentados o número de contribuidores, as linguagens utilizadas no

[8] https://play.google.com/store/apps/details?id=code.name.monkey.retromusic
[9] https://play.google.com/store/apps/details?id=it.feio.android.omninotes
[10] https://play.google.com/store/apps/details?id=org.wikipedia

desenvolvimento da aplicação e a quantidade de releases, extraídos do GitHub, bem como a quantidade de linhas de código relatada pela ferramenta SonarCloud. No caso específico das aplicações Aegis Authenticator e Amaze File Manager é importante observar que ambas foram categorizadas pela loja de aplicativos como ferramentas. No entanto, por terem utilidades distintas, foram classificadas neste estudo como sendo de segurança e de utilidades, respectivamente.

Tabela 3 – Características das aplicações selecionadas

	Aegis	**Amaze**	**Omni Notes**	**Retro**	**Wikipédia**
Avaliação	4,7	3,6	4,4	4,6	4,6
Avaliações	3,4 mil	12,4 mil	3,87 mil	74,4 mil	689 mil
Categoria	Segurança	Utilidades	Produtividade	Mídia	Livros
Contribuidores	50	137	35	61	114
Downloads	100 mil	1 mi	100 mil	1 mi	50 mi
Linguagens	Java	Java/Kotlin	Java	Kotlin	Kotlin
Linhas	41 mil	76 mil	36 mil	89 mil	188 mil
Releases	55	65	49	32	4

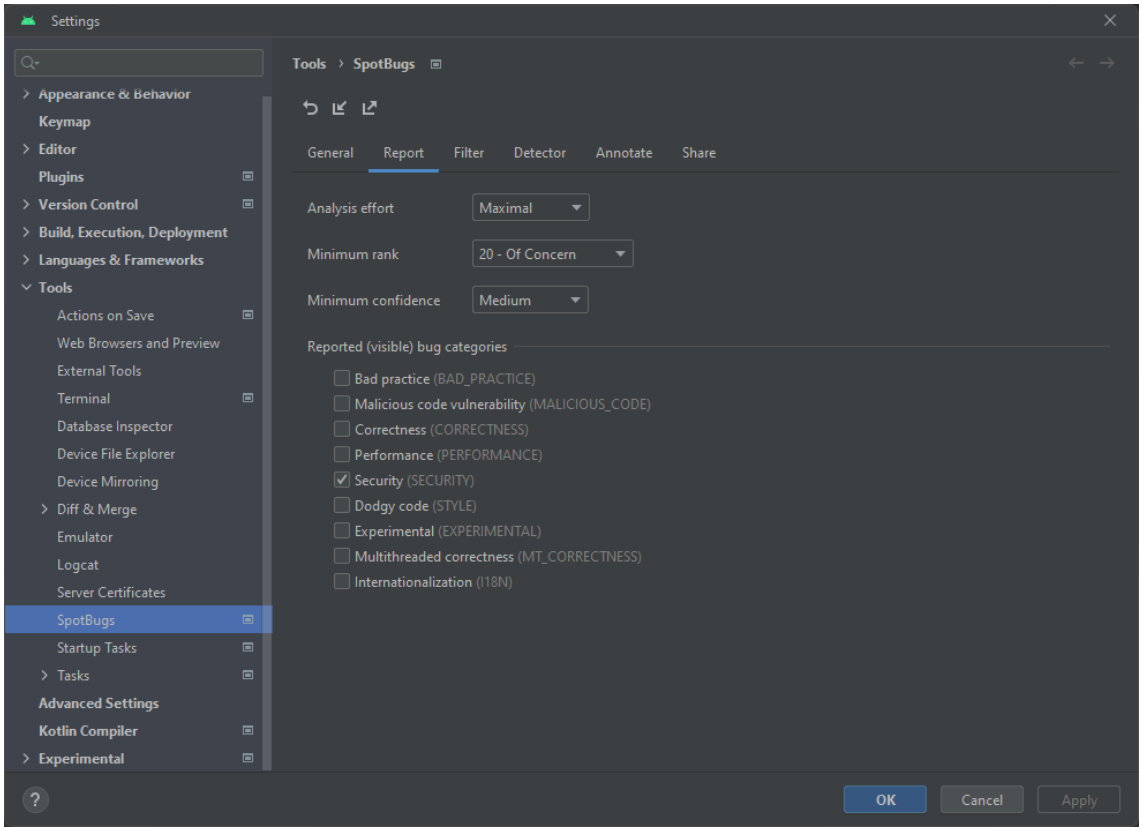
5 Execução dos Testes Estáticos nas Aplicações Selecionadas

Com o objetivo de obter os relatórios de análise nas aplicações selecionadas e as outras métricas previamente definidas, cada uma das cinco aplicações foi analisada por cada uma das quatro ferramentas selecionadas. Neste capítulo são detalhadas as configurações utilizadas para os testes e os resultados obtidos.

5.1 Configuração das Ferramentas

Para a execução do FindSecBugs, integrado ao SpotBugs, foram definidas algumas configurações do plugin no Android Studio, IDE utilizada para a execução da ferramenta de análise. A configuração "Esforço de análise" foi definida para o máximo, a fim de aumentar o rigor da análise, permitindo assim que sejam examinadas também classes referenciadas pelo aplicativo e métodos muito longos. As configurações de "Classificação mínima" e "Confiança mínima" foram mantidas nos valores padrão. As categorias foram definidas para considerar apenas as falhas de segurança, pois o foco do estudo está nesse tipo de vulnerabilidades. É importante destacar que essa última configuração reduz significativamente o tempo da análise e que essas são as configurações recomendadas na documentação da ferramenta.

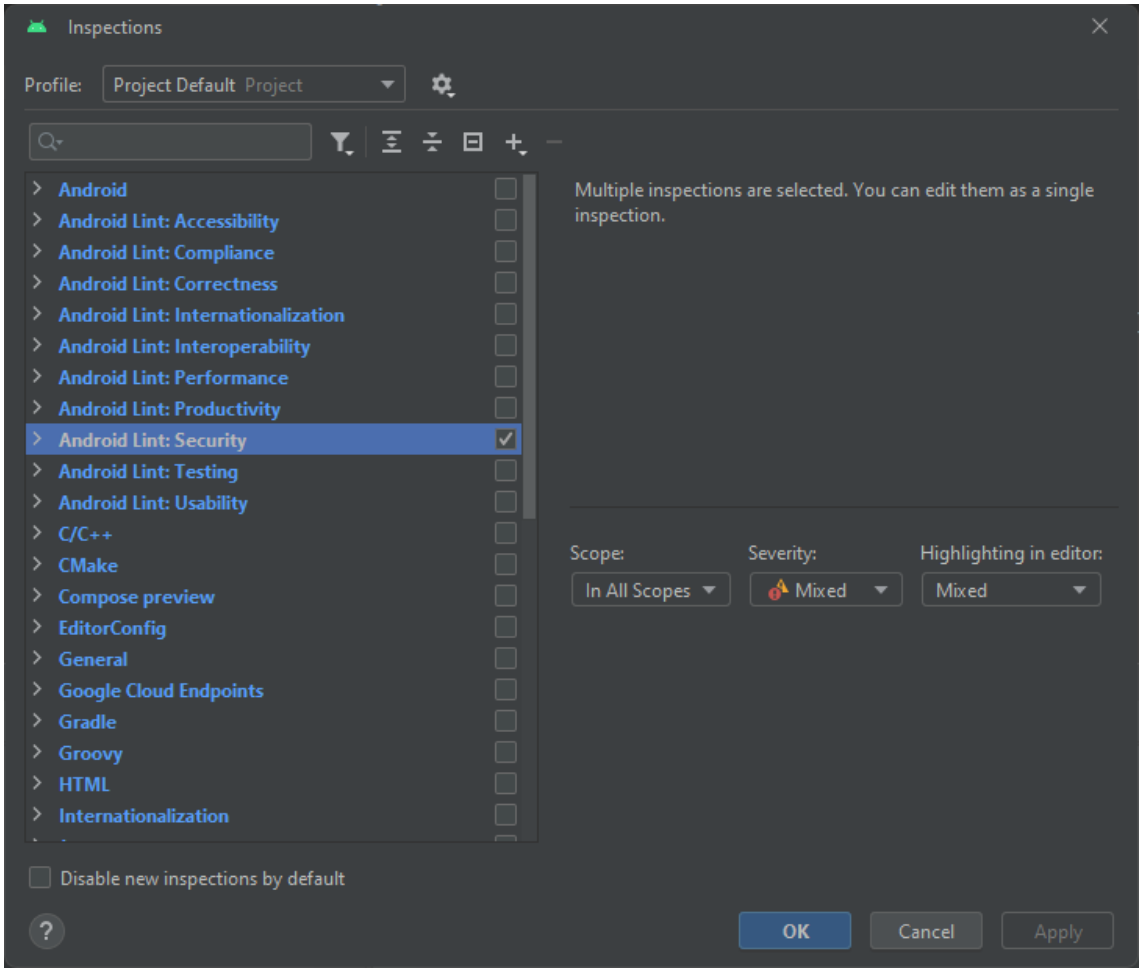
Figura 12 – Configurações do SpotBugs e FindSecBugs



Fonte: Elaboração Própria

O Android Lint, também executado pelo Android Studio, foi configurado para detectar apenas vulnerabilidades de segurança, assim como o FindSecBugs, otimizando a performance da análise. As demais configurações foram mantidas em seus valores padrão, a fim de considerar todo o escopo do projeto em análise e os diversos graus de severidade das vulnerabilidades.

Figura 13 – Configurações do Android Lint



Fonte: Elaboração Própria

Embora o SonarCloud realize uma verificação automática do projeto para detecção das vulnerabilidades, bugs e outros dados, a compilação do projeto antes da análise permite obter resultados mais completos por parte da ferramenta. Para a utilização do SonarCloud foi necessário alterar o arquivo build.gradle de cada aplicação para incluir o plugin do SonarCloud e as propriedades do plugin para a correta autenticação do projeto na ferramenta. Em seguida, o projeto foi compilado e analisado pelo SonarCloud por meio de comandos gradle, que automaticamente realiza o upload dos resultados para visualização pela API do SonarCloud ao final da análise. É possível configurar o Github Actions para realizar a análise de forma automática a cada atualização do repositório, porém para uma melhor medição do tempo de execução da análise ele foi executado na mesma máquina local que as outras ferramentas.

O MobSF possui um website onde pode ser feito o upload do código ou do APK do aplicativo a ser analisado, porém a ferramenta foi configurada na mesma máquina que as demais ferramentas para uma comparação mais eficaz do tempo de execução das análises. Foram executadas análises a partir do código fonte em formato ZIP. Essa decisão foi mo-

tivada pela razão de que, apesar da ferramenta apresentar resultados diferentes ao utilizar o aplicativo empacotado em formato APK, as referências para o código direcionavam para o código pré-compilado, e não para o original, tornando o processo de análise de falsos positivos ineficiente.

5.2 Versões das Aplicações

Para a execução dos testes estáticos foram usadas as versões das aplicações apresentadas na Tabela 4, com a release de cada aplicação obtida diretamente do GitHub.

Tabela 4 – Versões das aplicações

Aplicação	**Versão de Release**
Aegis Authenticator	v2.2.2
Amaze File Manager	v3.8.4
Wikipédia	r-2.7.50452-r-2023-09-06
Omni Notes	6.2.8
Retro Music Player	Release v6.1.0 - Production

5.3 Versões das Ferramentas

As ferramentas foram utilizadas nas versões apresentadas na Tabela 5. É importante notar que a versão utilizada no Android Lint depende da versão do Android Gradle Plugin, que pode variar de projeto para projeto. Além disso, é relevante destacar que o SpotBugs possui um versionamento diferente do FindSecBugs, pois este é uma extensão do SpotBugs. Portanto, foram especificadas as versões de ambos.

Tabela 5 – Versões das ferramentas

Ferramenta	**Versão**
Android Lint	Android Gradle Plugin
SpotBugs / FindSecBugs	1.2.5 / 1.12.0
MobSF	v3.7.8
SonarCloud	4.3.1.3277

5.4 Tempos de Execução

Para a obtenção de resultados mais precisos, todos os tempos foram obtidos por meio da execução em uma mesma máquina. As especificações da máquina podem ser observadas a seguir:

- *Intel Core* i7-8750H;

- 16 GB RAM;
- *SSD NVME* 500 GB;
- *Windows* 11.

A Tabela 6 apresenta os tempos de análise por parte das ferramentas para cada aplicação.

Tabela 6 – Tempos de execução das análises

	Android Lint	**FindSecBugs**	**MobSF**	**SonarCloud**
Aegis Authenticator	10 s	13 s	7 s	2 min 3 s
Amaze File Manager	1 min 34 s	23 s	12 s	4 min 4 s
Retro Music Player	1 min 48 s	11 s	8 s	2 min 48 s
Omni Notes	18 s	11 s	8 s	2 min 37 s
Wikipédia	3 min 25 s	9 s	9 s	4 min 26 s

5.5 Quantidade de Vulnerabilidades

A Tabela 7 apresenta a quantidade de vulnerabilidades detectadas por cada ferramenta indicada no relatório de execução de análise estática para cada uma das aplicações da amostra.

Visando o comparativo das ferramentas, foram consideradas apenas vulnerabilidades cuja severidade estava marcada como sendo de confiança média ou alta, visto que essa é a configuração padrão do FindSecBugs.

No Android Lint foram consideradas as vulnerabilidades marcadas com status de atenção e erro, correspondendo, respectivamente, às categorias média e alta de severidade. Esta é a configuração padrão da ferramenta.

O MobSF classifica as vulnerabilidades como sendo de segurança, info, aviso e alta. Não foram consideradas as de segurança, pois se trata de aspectos positivos, e não de falhas. As classificadas como info também não foram consideradas, pois há um grau de confiança mais baixo, além de não serem mapeadas pela própria ferramenta para o OWASP Mobile Top 10, ao contrário do que é feito com as de maior severidade. Assim, somente foram consideradas aquelas categorizadas como aviso, que seriam as médias, e como alta.

Em relação ao SonarCloud, foram consideradas todas as vulnerabilidades apresentadas, visto que estavam marcadas com a severidade média ou alta. Além disso, não foram considerados os *hotspots* indicados no relatório da análise estática, pois a própria ferramenta indica que são pontos que podem ou não ser uma vulnerabilidade, o que sugere um nível de confiança menor por parte da ferramenta.

Tabela 7 – Quantidade de vulnerabilidades

	Android Lint	**FindSecBugs**	**MobSF**	**SonarCloud**
Aegis Authenticator	0	23	40	6
Amaze File Manager	9	122	110	15
Omni Notes	4	33	39	6
Retro Music Player	1	11	33	3
Wikipédia	8	0	22	1

6 Análise de Falsos Positivos

Com o objetivo de identificar quais vulnerabilidades detectadas pelas ferramentas não se tratavam de uma vulnerabilidade real para a aplicação, os autores, individualmente, analisaram os relatórios obtidos com a execução da análise estática realizada pelas ferramentas.

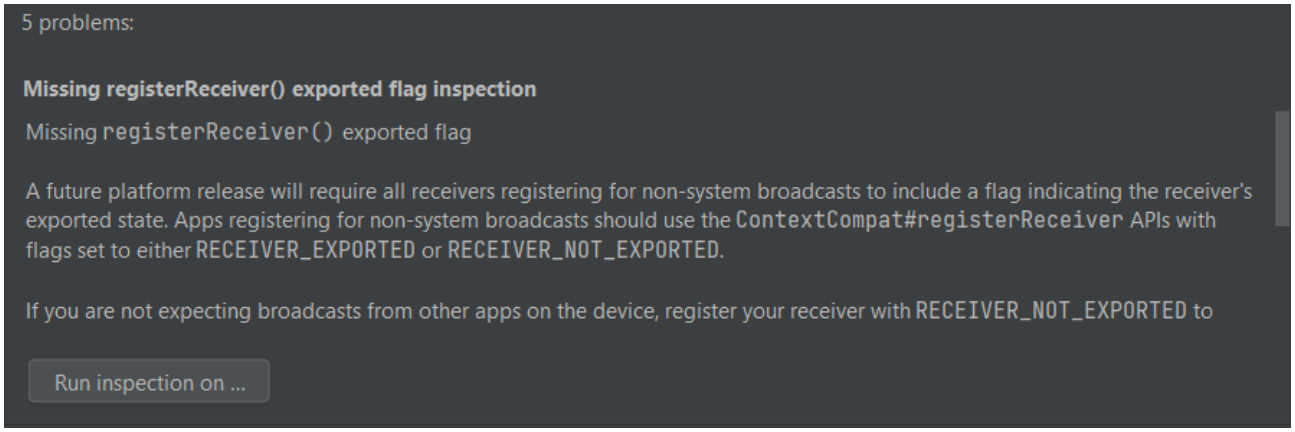
Cada vulnerabilidade foi analisada de forma isolada e seguindo três passos fundamentais: primeiro, a compreensão das vulnerabilidades apresentadas no relatório gerado pela ferramenta. Segundo, a localização da vulnerabilidade no código fonte da aplicação. Terceiro, a análise quanto a falso positivo ou verdadeiro positivo. As seções seguintes apresentam um maior detalhamento sobre cada passo realizado.

6.1 Passo 1: Compreensão da Vulnerabilidade

Para o primeiro passo, realizou-se a leitura das informações contidas nos relatórios gerados pelas ferramentas a respeito da vulnerabilidade indicada. Cada ferramenta apresentou um relatório contendo informações diferentes.

Com respeito ao Android Lint, quando o relatório é acessado pela própria ferramenta, é possível ver alguns detalhes adicionais, conforme observado na Figura 14. No entanto, ao exportar o relatório para uma versão em formato html, esse recurso não é mantido, mostrando apenas as vulnerabilidades e o nome do arquivo onde essas se encontram.

Figura 14 – Exemplo de informações presentes no relatório do Android Lint



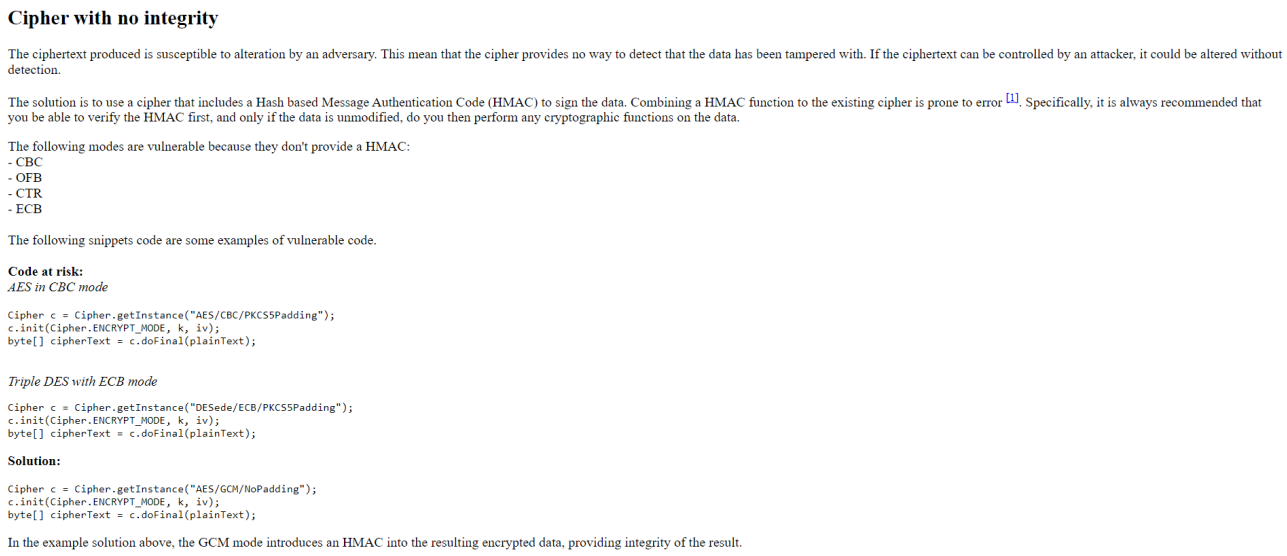
Fonte: Elaboração Própria

Com respeito ao FindSecBugs, tanto o relatório sendo observado na própria ferramenta, quanto a versão em html, apresentam informações adicionais a respeito das

vulnerabilidades encontradas pela ferramenta, como, por exemplo, a descrição da vulnerabilidade, exemplos de códigos que apresentam aquele risco e exemplos de como solucionar o problema.

Na Figura 15 é possível observar um exemplo de detalhamento para a vulnerabilidade "*Cipher with no integrity*", presente na versão html do relatório do FindSecBugs.

Figura 15 – Detalhamento de vulnerabilidade no relatório exportado do FindSecBugs



Cipher with no integrity

The ciphertext produced is susceptible to alteration by an adversary. This mean that the cipher provides no way to detect that the data has been tampered with. If the ciphertext can be controlled by an attacker, it could be altered without detection.

The solution is to use a cipher that includes a Hash based Message Authentication Code (HMAC) to sign the data. Combining a HMAC function to the existing cipher is prone to error [1]. Specifically, it is always recommended that you be able to verify the HMAC first, and only if the data is unmodified, do you then perform any cryptographic functions on the data.

The following modes are vulnerable because they don't provide a HMAC:
- CBC
- OFB
- CTR
- ECB

The following snippets code are some examples of vulnerable code.

Code at risk:
AES in CBC mode

```
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
c.init(Cipher.ENCRYPT_MODE, k, iv);
byte[] cipherText = c.doFinal(plainText);
```

Triple DES with ECB mode

```
Cipher c = Cipher.getInstance("DESede/ECB/PKCS5Padding");
c.init(Cipher.ENCRYPT_MODE, k, iv);
byte[] cipherText = c.doFinal(plainText);
```

Solution:

```
Cipher c = Cipher.getInstance("AES/GCM/NoPadding");
c.init(Cipher.ENCRYPT_MODE, k, iv);
byte[] cipherText = c.doFinal(plainText);
```

In the example solution above, the GCM mode introduces an HMAC into the resulting encrypted data, providing integrity of the result.

Fonte: Elaboração Própria

A ferramenta SonarCloud, por sua vez, apresenta na maioria dos casos cinco seções importantes para cada vulnerabilidade indicada em seu relatório. A primeira seção apresenta o trecho do código da aplicação analisada, indicando onde está a vulnerabilidade. A segunda seção é nomeada como "Por que isso é um problema?", apresentando o motivo da detecção da vulnerabilidade e o potencial impacto que tal vulnerabilidade pode causar. A terceira seção não está presente em todas as vulnerabilidades detectadas e diz respeito à solução do problema. Essa seção apresenta um exemplo da vulnerabilidade e como o problema do exemplo apresentado pode ser corrigido. A quarta seção é referente ao rastreamento das atividades realizadas sobre a vulnerabilidade, tal como a data de criação e a data em que o problema foi marcado como resolvido. Por fim, a quinta seção é a de mais informações sobre a vulnerabilidade, apresentando links úteis e, em alguns casos, o mapeamento para o CWE e para o OWASP Mobile Top 10 de 2016.

O relatório gerado pela ferramenta MobSF contém, para cada um dos tipos de vulnerabilidade detectadas, a severidade, o título, o mapeamento para o CWE e o OWASP Mobile Top 10 associados, além do caminho do arquivo onde a vulnerabilidade foi encontrada. É possível obter mais informações ao clicar no link presente no título da vulnerabilidade, o qual redireciona para um arquivo em Markdown no repositório owasp-mstg, mantido pela equipe do MobSF. O objetivo desse arquivo apontado pelo link é detalhar

a vulnerabilidade detectada.

6.2 Passo 2: Localização da Vulnerabilidade no Código Fonte

Por serem ferramentas executadas no próprio Android Studio, o Android Lint e o FindSecBugs já apontam diretamente para o trecho de código onde a vulnerabilidade foi encontrada. No entanto, ao se utilizar a versão do relatório em html exportada por essas ferramentas, esse recurso não está disponível. Além disso, o relatório gerado pelo Android Lint na versão html apresenta apenas o nome do arquivo onde a vulnerabilidade foi encontrada, dificultando a atividade de identificação da vulnerabilidade no código fonte. Por outro lado, o FindSecBugs apresenta o caminho para o arquivo e a linha de código onde a vulnerabilidade foi detectada.

Para o segundo passo, o SonarCloud apresenta um recurso útil em seu relatório, pois permite a visualização do próprio arquivo, indicando por meio de comentário onde está a vulnerabilidade. Esse recurso se configura como uma facilidade apresentada pela ferramenta, pois, mesmo o relatório da ferramenta sendo apresentado em sua plataforma em nuvem, elimina a necessidade do testador abrir o código fonte para procurar a vulnerabilidade.

No que diz respeito à atividade de localização da vulnerabilidade no código fonte, o relatório gerado pelo MobSF conta com um link para cada arquivo onde a vulnerabilidade foi encontrada, visto que a ferramenta tem suporte para abrir o arquivo no navegador. No entanto, esse suporte é limitado para arquivos escritos em Java, mesmo que a ferramenta realize análise em códigos escritos em Kotlin. Ao abrir o arquivo, direciona-se automaticamente às linhas que estão destacadas, indicando a localização da vulnerabilidade detectada.

6.3 Passo 3: Verificar Falsos Positivos

O terceiro passo foi realizado por se analisar o trecho de código apontado como vulnerável pelas ferramentas. A conclusão em relação a ser ou não um falso positivo foi tomada com o auxílio de pesquisas sobre a vulnerabilidade indicada no relatório. Em alguns casos as próprias ferramentas apresentavam exemplos a respeito da vulnerabilidade. Para esses casos, os exemplos fornecidos pela ferramenta também eram levados em consideração para a identificação de falsos positivos.

Após a realização de todos os passos apresentados, obteve-se duas listas de falsos positivos, uma para cada autor. As Tabelas 8 e 9 apresentam a quantidade de falsos positivos para cada uma das ferramentas em relação a cada uma das aplicações, identificados por Kleidson Alves e Lucas Rodrigues, respectivamente.

Tabela 8 – Quantidade de falsos positivos encontrados por Kleidson Alves

	Android Lint	**FindSecBugs**	**MobSF**	**SonarCloud**
Aegis Authenticator	0	13	25	0
Amaze File Manager	0	94	84	4
Omni Notes	2	22	19	0
Retro Music Player	0	9	7	1
Wikipédia	0	0	10	1

Tabela 9 – Quantidade de falsos positivos encontrados por Lucas Rodrigues

	Android Lint	**FindSecBugs**	**MobSF**	**SonarCloud**
Aegis Authenticator	0	14	12	2
Amaze File Manager	0	95	16	0
Omni Notes	0	25	4	0
Retro Music Player	0	10	27	0
Wikipédia	0	0	11	0

As Tabelas 8 e 9 indicam uma diferença pequena nos resultados das análises de falsos positivos feitas pelos autores, estando a maioria em um intervalo de um a cinco. No entanto, o objetivo esperado para essa etapa era de uma única lista. Sendo assim, houve a necessidade de unificar a lista de falsos positivos encontrados pelos autores.

Para dar continuidade à pesquisa, os autores apresentaram argumentos para justificar as vulnerabilidades que identificaram como falsos positivos. Após isso, os autores realizavam uma nova análise, agora em conjunto, para determinar a validade dos argumentos. Por fim, chegou-se a um consenso no que diz respeito aos falsos positivos, obtendo a quantidade de falsos positivos para cada ferramenta em relação a cada aplicação, conforme apresentado na Tabela 10.

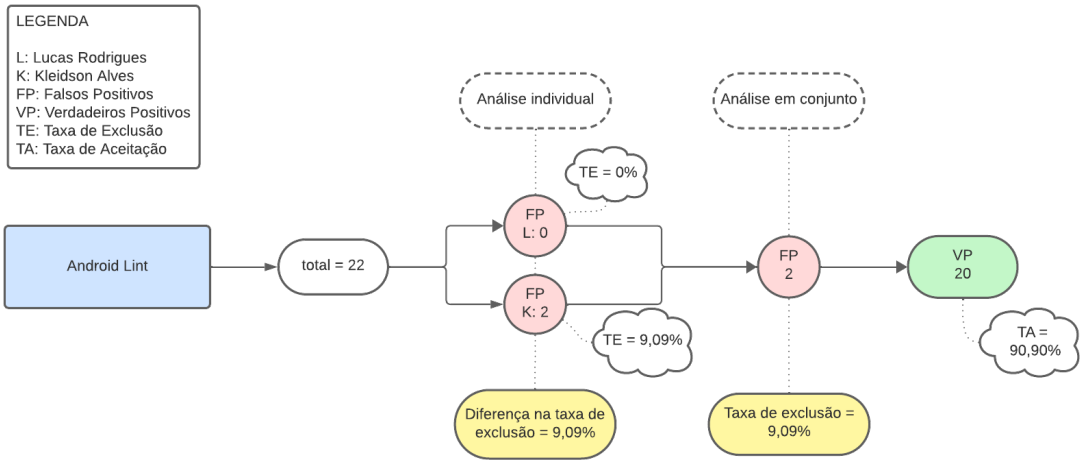
Tabela 10 – Quantidade de falsos positivos

	Android Lint	**FindSecBugs**	**MobSF**	**SonarCloud**
Aegis Authenticator	0	14	22	2
Amaze File Manager	0	95	86	4
Omni Notes	2	25	23	0
Retro Music Player	0	10	30	1
Wikipédia	0	0	14	1

As imagens a seguir mostram o processo de filtragem de falsos positivos para cada ferramenta. No início do processo é exibido o número total de vulnerabilidades identificadas pela ferramenta para toda a amostra de aplicações. Na etapa de análise individual são exibidas as quantidades de falsos positivos e as taxas de exclusão de vulnerabilidades, com base na análise feita por cada pesquisador, além da diferença entre a porcentagem encontrada por cada um. Em seguida, na etapa de análise em conjunto, é apresentada a

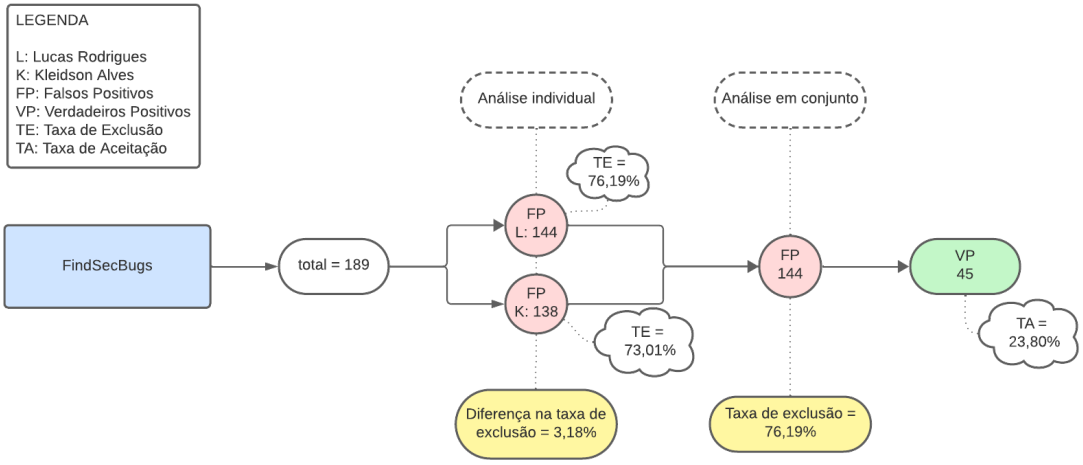
quantidade de falsos positivos da lista unificada e a taxa de exclusão de vulnerabilidades com relação ao total inicialmente identificado pela ferramenta. Por fim, há a quantidade de verdadeiros positivos e a taxa de aceitação, que corresponde à porcentagem de verdadeiros positivos em relação ao total de vulnerabilidades inicial.

Figura 16 – Processo de filtragem do Android Lint



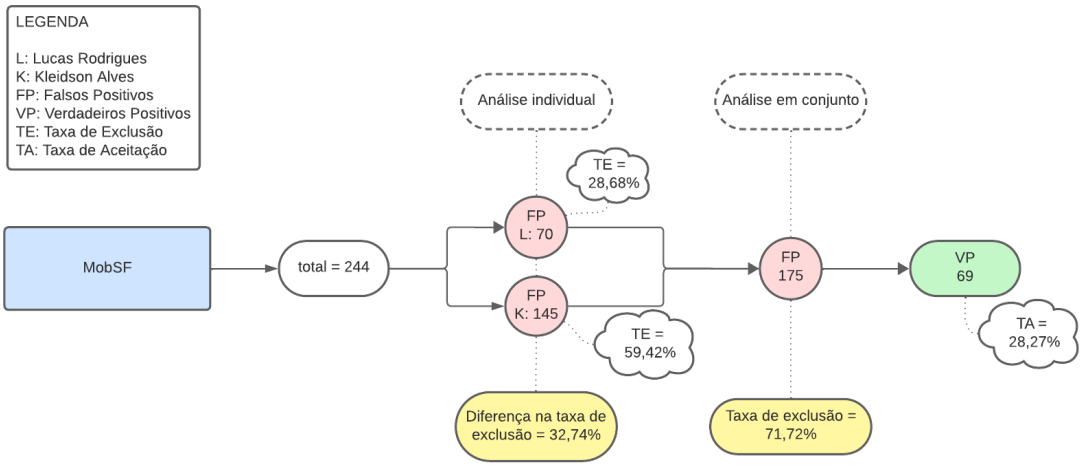
Fonte: Elaboração Própria

Figura 17 – Processo de filtragem do FindSecBugs



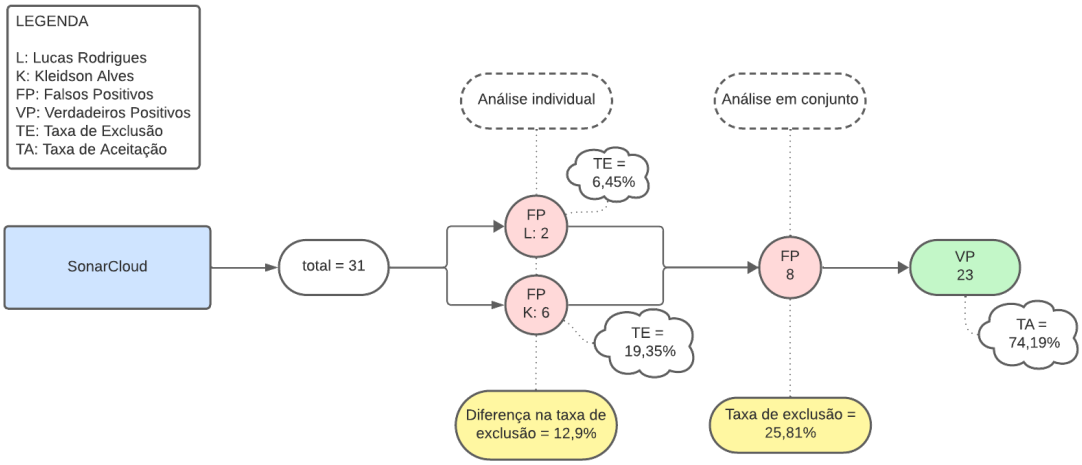
Fonte: Elaboração Própria

Figura 18 – Processo de filtragem do MobSF



Fonte: Elaboração Própria

Figura 19 – Processo de filtragem do SonarCloud



Fonte: Elaboração Própria

Durante o processo de verificação das vulnerabilidades, identificou-se que os falsos positivos do Android Lint estão relacionados à duplicação de uma vulnerabilidade apresentada no relatório da análise estática. Sendo assim, para identificar como falso positivo, foi levado em consideração que o relatório indica uma quantidade de vulnerabilidades maior do que realmente a aplicação possui.

A maioria dos falsos positivos detectados pelo FindSecBugs está relacionada à leitura de um arquivo cujo nome é fornecido por um usuário, porém isso não se materializa nas aplicações da amostra como sendo uma vulnerabilidade real a ser explorada, uma vez que a entrada feita pelo usuário não é diretamente inserida no código. Outra vulnerabilidade identificada com frequência pelo FindSecBugs está relacionada aos *Broadcast intents*. A vulnerabilidade indicada pela ferramenta é que as transmissões de dados po-

dem ser recebidas por aplicações maliciosas. Configura-se, porém, como um falso positivo, visto que a aplicação não transmite dados sensíveis e utiliza corretamente os métodos da classe *Intent*. Além disso, a ferramenta aponta o risco de injeção de SQL em algumas aplicações, alegando o uso de comando SQL fornecido pelo usuário. Porém, a construção do comando SQL utilizado nas aplicações é feita apenas em seus próprios códigos fonte.

Grande parte dos falsos positivos detectados pelo MobSF está relacionada com a vulnerabilidade de ocultar elementos da interface. Isso pode representar um risco para a aplicação quando o elemento oculto contém dados sensíveis, que podem ser obtidos e expostos por atacantes. No entanto, muitos dos casos apontados pela ferramenta não se encaixam como uma vulnerabilidade para a aplicação, pois os elementos escondidos não apresentam informações sensíveis e são, por exemplo, botões, barras de progresso ou componentes de tela para tutoriais.

Outra vulnerabilidade comum identificada pelo MobSF diz respeito à presença de senhas e chaves secretas no código fonte. Os trechos de código indicados pela ferramenta, no entanto, consistiam no uso da palavra "key" para indicar o atributo de alguma preferência do usuário ou de uma configuração da aplicação, não necessariamente representando em todos os casos uma exposição real de dados sensíveis.

Assim como o FindSecBugs, o MobSF apresentou algumas ocorrências de vulnerabilidade em relação a injeção de SQL e, pelo mesmo motivo, foi identificado como falso positivo.

Com respeito ao SonarCloud, os falsos positivos detectados estão relacionados à análise feita pela ferramenta sobre o arquivo Android Manifest das aplicações, onde a ferramenta indica a necessidade de permissões para exportações de componentes, mas os componentes são configurados para visualização apenas, não exigindo permissões adicionais.

7 Mapeamento das Vulnerabilidades

Com a execução das etapas anteriores, obteve-se um conjunto de vulnerabilidades reais a partir do relatório de cada ferramenta para cada aplicação selecionada. Com o objetivo de identificar o CWE e a categoria do OWASP Mobile Top 10 associados, realizou-se o mapeamento das vulnerabilidades.

As ferramentas MobSF e SonarCloud, na grande maioria dos casos, realizaram o mapeamento das vulnerabilidades encontradas para o CWE e para o OWASP Mobile Top 10. Por outro lado, o Android Lint não apresentava esse mapeamento para nenhuma das vulnerabilidades detectadas, enquanto o FindSecBugs o fazia apenas para o CWE na maior parte dos casos. Sendo assim, mostrou-se necessário os próprios autores mapearem as vulnerabilidades. Esta seção apresenta como esse processo foi realizado.

As vulnerabilidades relatadas foram inseridas em uma planilha para realizar o mapeamento. Com o objetivo de identificar o CWE associado a cada vulnerabilidade encontrada, principalmente pelas ferramentas Android Lint e FindSecBugs, verificava-se a semelhança da vulnerabilidade com outras que já haviam sido mapeadas automaticamente por outra ferramenta. Para isso, observava-se a descrição da vulnerabilidade e o trecho de código indicado pela ferramenta.

Com relação às vulnerabilidades que não haviam sido mapeadas para o CWE por nenhuma outra ferramenta, foi necessário usar a busca manual no catálogo do CWE, que por sua vez foram mapeadas para o OWASP Mobile Top 10.

Houve necessidade de mapear manualmente dois tipos de vulnerabilidade encontradas pelo Android Lint na aplicação Amaze File Manager para o CWE. A primeira delas está relacionada com uma mudança futura que será adotada na plataforma do Android, na qual será exigido que se indique se um receptor será ou não exportado, a fim de garantir a segurança desse receptor. Essa vulnerabilidade foi mapeada para o CWE 925, que se refere à falta de verificação se um transmissor que recebe um *intent* vem de uma fonte autorizada. Por sua vez, esse CWE foi mapeado para a categoria M1 do OWASP (Uso Indevido da Plataforma), uma vez que se trata de uma fraqueza no uso do controle de segurança da plataforma. A segunda vulnerabilidade se trata do nome de uma permissão que não segue a convenção, podendo potencialmente causar um desentendimento com relação ao proprietário e a intenção da permissão. Essa vulnerabilidade foi mapeada para o CWE 1099, que se refere ao uso inconsistente de nomes de identificadores. Essa vulnerabilidade também foi mapeada para a categoria M1 do OWASP, pois dentre as suas características está a violação de convenções e boas práticas.

Na aplicação Wikipédia, para a mesma ferramenta, foi necessário mapear manual-

mente mais duas vulnerabilidades. A primeira identificou que o aplicativo não especificou um nome de classe de componente totalmente qualificado, permitindo que um aplicativo malicioso intercepte o *intent* no lugar do aplicativo pretendido. Essa vulnerabilidade foi mapeada para o CWE 927, que diz que, como um *intent* implícito não especifica um aplicativo específico para receber os dados, qualquer aplicativo pode processar o *intent*. Essa vulnerabilidade foi mapeada para as categorias do OWASP M1, em função do uso inadequado do controle de segurança da plataforma, e M6 (Autorização Insegura), pois poderia explorar a vulnerabilidade relacionada à autorização para obter acesso ao *intent*. A segunda vulnerabilidade encontrada nessa aplicação se trata do uso de uma propriedade obsoleta na aplicação. Essa vulnerabilidade foi mapeada para o CWE 477 (uso de função obsoleta) e para o M1 do OWASP, por se tratar do uso inadequado de permissões da plataforma.

Para todas as aplicações, o MobSF identificou uma vulnerabilidade relacionada ao arquivo Android Manifest. A vulnerabilidade estava relacionada com a permissão de fazer backup por parte de um controle não autorizado. Essa vulnerabilidade foi mapeada para o CWE 530, que se refere à exposição do arquivo de backup para um controle não autorizado. Esse CWE, por sua vez, está relacionado à categoria M1 do OWASP Mobile Top 10.

No que diz respeito ao FindSecBugs, foi necessário apenas o mapeamento do CWE, indicado pela ferramenta, para as categorias do OWASP Mobile Top 10. Por exemplo, para a aplicação Retro Music Player, foi detectada a vulnerabilidade relacionada ao CWE 918, que diz respeito à falsificação de solicitação do lado do servidor. Esse CWE foi mapeado para a categoria M3 do OWASP Mobile Top 10, comunicação insegura.

8 Análise dos Resultados

Este capítulo tem como objetivo apresentar a análise realizada sobre os resultados obtidos nas etapas anteriores e como está relacionada com as perguntas de pesquisa. Para isso, foram criados diagramas e tabelas que apresentam uma melhor visualização dos dados, os quais estão relacionados aos critérios para comparação das quatro ferramentas selecionadas e às vulnerabilidades mais comuns presentes na amostra de aplicações.

No que diz respeito à primeira pergunta de pesquisa (Q1), foram levados em consideração três critérios importantes para a comparação das ferramentas. Primeiro, a precisão da análise realizada, levando em consideração a quantidade total de vulnerabilidades detectadas, a quantidade de falsos positivos e a quantidade de verdadeiros positivos. Segundo, a quantidade de vulnerabilidades de diferentes categorias identificadas. Terceiro, o relato dos pesquisadores sobre a experiência de uso das ferramentas.

A segunda pergunta de pesquisa (Q2) está relacionada com as vulnerabilidades mais comuns presentes nas aplicações da amostra. Sendo assim, mostra-se necessário apenas os dados relacionados ao índice de ocorrência de cada categoria de vulnerabilidade.

Este capítulo está organizado de forma a apresentar a análise da precisão das ferramentas, a análise da identificação das vulnerabilidades do OWASP Mobile Top 10, a análise dos relatos feitos pelos pesquisadores a respeito do uso das ferramentas e a análise das vulnerabilidades mais comuns presentes na amostra de aplicações.

8.1 Precisão

A quantidade de falsos positivos encontrados é uma métrica importante que deve ser levada em consideração como um dos critérios para responder a Q1, pois tem impacto no indicativo de precisão da análise realizada pela ferramenta. Nesse respeito, a Tabela 11 apresenta o total de falsos positivos detectados por cada ferramenta. Esse resultado pode ser obtido através do somatório da quantidade de falsos positivos encontrados por cada ferramenta para cada aplicação da amostra, conforme apresentado na Tabela 10.

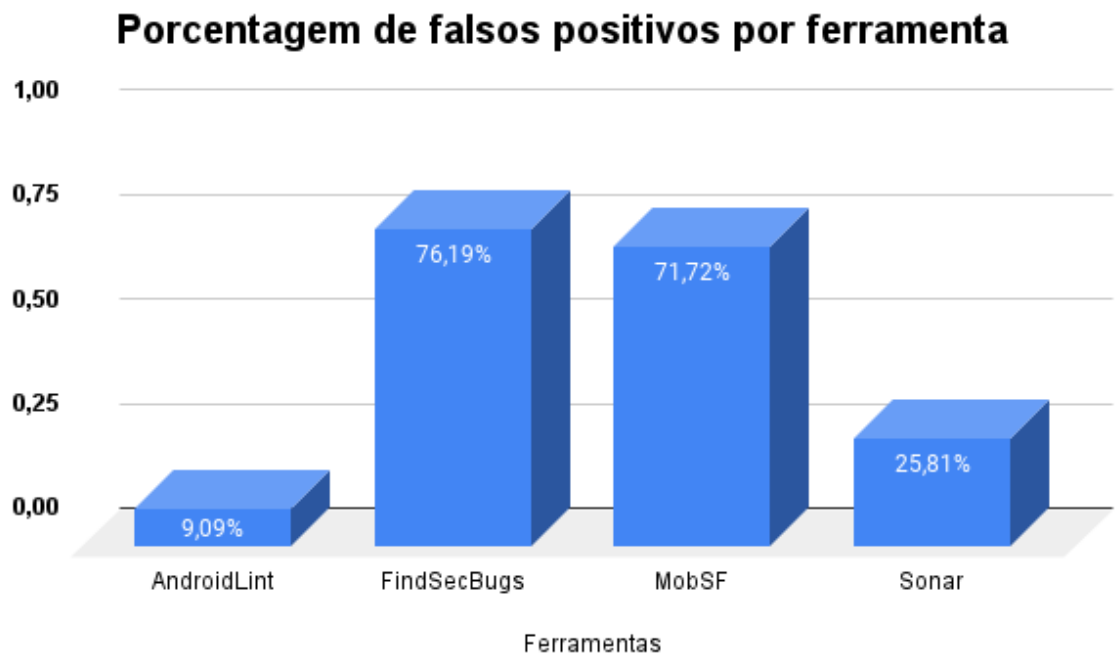
Tabela 11 – Total de falsos positivos por ferramenta

Ferramenta	**Total de falsos positivos**
Android Lint	2
FindSecBugs	144
MobSF	175
SonarCloud	8

Observa-se na Tabela 11 que o Android Lint apresentou apenas dois falsos positivos para toda a amostra. Trata-se de uma quantidade 72 e 87,5 vezes menor do que a quantidade de falsos positivos identificados pelo FindSecBugs e pelo MobSF, respectivamente. O SonarCloud apresenta o segundo melhor resultado, tendo a quantidade total de falsos positivos detectados cerca de 18 e 22 vezes menor em relação ao FindSecBugs e ao MobSF, respectivamente.

A Figura 20 indica a porcentagem que o total de falsos positivos encontrados por cada ferramenta - mostrado na Tabela 11 - corresponde em relação a quantidade de vulnerabilidades detectadas por ela em todas as aplicações da amostra. Quanto maior a porcentagem, menor é o desempenho da ferramenta quanto ao critério de identificação de falsos positivos.

Figura 20 – Porcentagem de falsos positivos por ferramenta



Fonte: Elaboração Própria

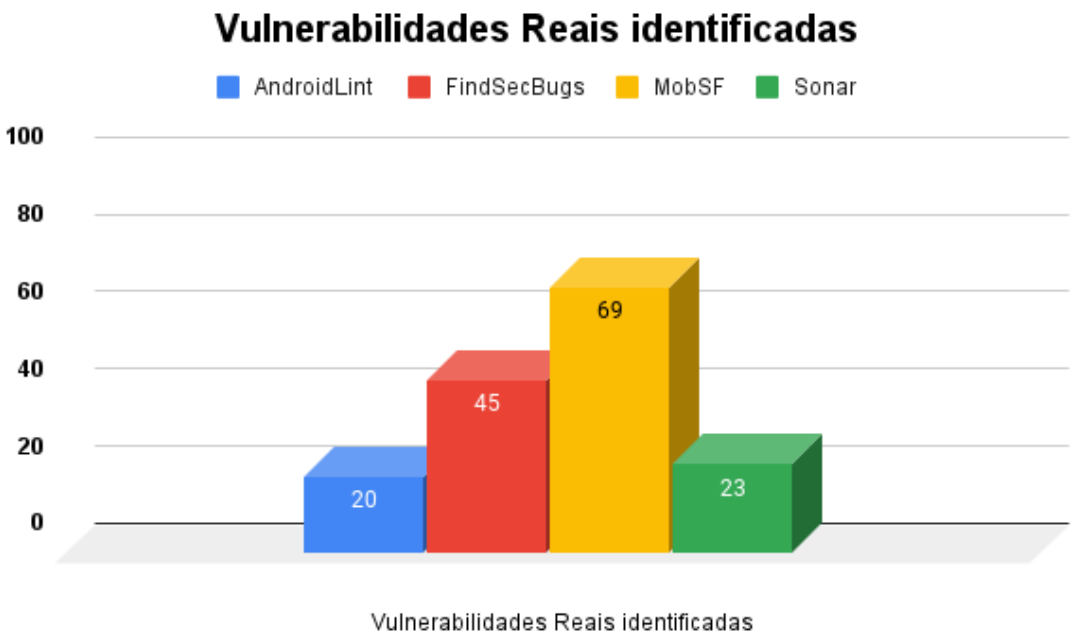
De acordo com o gráfico da Figura 20, as ferramentas FindSecBugs e MobSF apresentaram uma alta densidade de falsos positivos em sua análise, visto que esses representam mais de 70% das vulnerabilidades encontradas por essas ferramentas. O gráfico também indica uma porcentagem consideravelmente inferior de falsos positivos identificados pelo Android Lint em relação às demais ferramentas, sendo mais de 10% a menos em relação ao SonarCloud, a segunda ferramenta com menor porcentagem de falsos positivos. Além disso, o gráfico mostra que os falsos positivos encontrados pelo SonarCloud correspondem a pouco mais de $\frac{1}{4}$ das vulnerabilidades encontradas na execução da análise estática. Trata-se de uma densidade de falsos positivos consideravelmente inferior em

relação às ferramentas MobSF e FindSecBugs.

É possível observar na Tabela 11 que o MobSF foi a ferramenta que apresentou mais falsos positivos. No entanto, como mostra a Figura 20, não se trata da ferramenta com maior porcentagem de falsos positivos detectados. Isso é justificado pelo fato de que mesmo sendo a ferramenta que identificou mais falsos positivos, essa também é a ferramenta que encontrou mais vulnerabilidades na amostra de aplicações.

No que diz respeito à precisão da análise realizada pelas ferramentas, também se mostra necessário comparar a capacidade das ferramentas quanto a identificação de verdadeiros positivos. A Figura 21 apresenta o gráfico referente à quantidade de vulnerabilidades reais encontradas por cada uma das ferramentas ao se executar a análise estática em todas as aplicações da amostra.

Figura 21 – Vulnerabilidades reais identificadas



Fonte: Elaboração Própria

Ao se comparar os gráficos apresentados nas Figura 20 e 21, percebe-se que, embora o MobSF e o FindSecBugs sejam as ferramentas que apresentam as maiores densidades de falsos positivos, também foram as que mais apresentaram vulnerabilidades reais. Conforme mostrado na Figura 21, essas ferramentas identificaram mais do que o dobro da quantidade de vulnerabilidades identificadas pelo Android Lint, a ferramenta com o menor desempenho quanto a identificação de vulnerabilidades.

O conhecimento a respeito da quantidade de verdadeiros positivos e de falsos positivos permite a realização do cálculo da precisão da análise realizada por cada ferramenta. De acordo com Antunes e Vieira (2015), a precisão é calculada da seguinte forma:

$$P = \frac{TP}{TP + FP}$$

Sendo P o valor da precisão, TP a quantidade de verdadeiros positivos e FP a quantidade de falsos positivos, a fórmula indica que a precisão pode ser obtida através da divisão da quantidade de verdadeiros positivos identificados pela soma das quantidades de verdadeiros positivos e falsos positivos. O valor da precisão encontra-se no intervalo entre 0 e 1. Dessa forma, quanto mais próximo de 1 for a precisão, mais alta ela é.

A Tabela 12 apresenta a precisão de cada ferramenta selecionada, calculada de acordo com a fórmula apresentada.

Tabela 12 – Precisão de análise das ferramentas

Ferramenta	**Precisão**
Android Lint	0,91
FindSecBugs	0,24
MobSF	0,28
SonarCloud	0,74

É possível observar que o Android Lint apresentou a melhor precisão quanto a detecção de vulnerabilidades nas aplicações. Por outro lado, o FindSecBugs apresentou a menor precisão em relação às demais ferramentas.

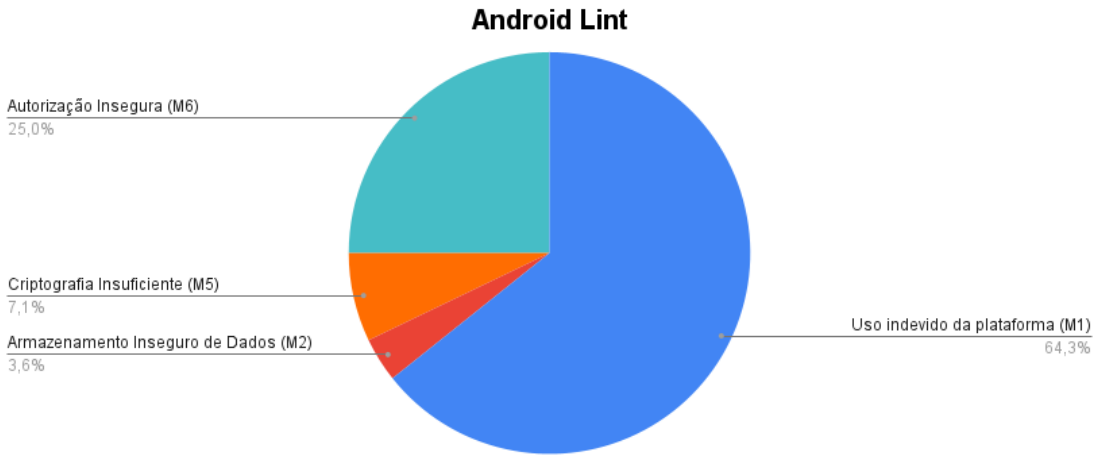
8.2 Vulnerabilidades OWASP Mobile Top 10

A capacidade de identificação de diferentes categorias de vulnerabilidade é uma característica importante para as ferramentas de análise estática. Mostra-se, então, necessário analisar, como um critério para comparação das ferramentas, a quantidade de categorias de vulnerabilidade que elas foram capazes de detectar. Para isso, utilizou-se como referência para categorias de vulnerabilidades a lista definitiva de 2016 do projeto OWASP Mobile Top 10[1].

Os gráficos presentes nas Figuras 22, 23, 24 e 25 apresentam a porcentagem das vulnerabilidades detectadas de cada categoria de vulnerabilidade da lista do OWASP Mobile Top 10 para, respectivamente, o Android Lint, o FindSecBugs, o MobSF e o SonarCloud.

[1] Top 10 Mobile Risks - Final List 2016: https://owasp.org/www-project-mobile-top-10/

Figura 22 – Porcentagem de ocorrências das vulnerabilidades do OWASP Mobile Top 10 no Android Lint

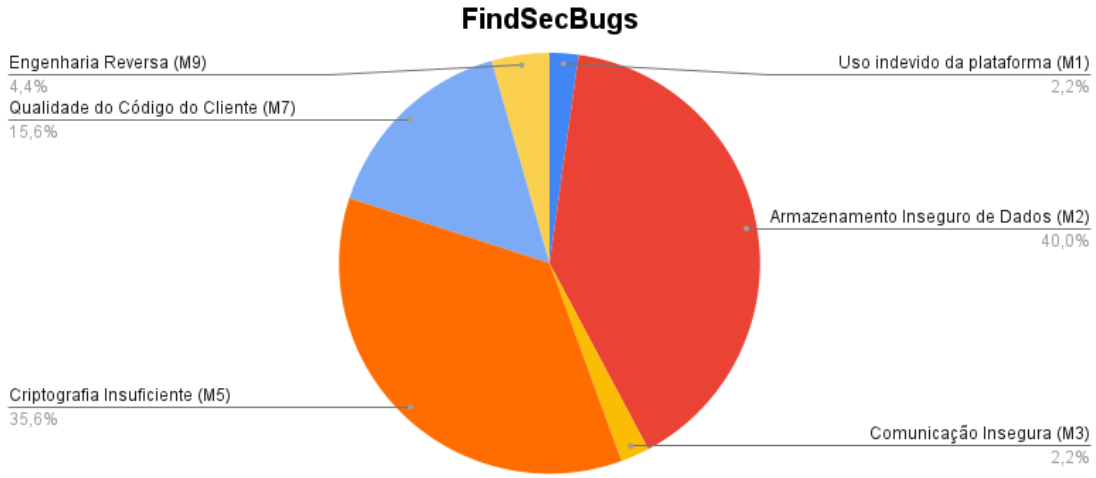


Fonte: Elaboração Própria

O Android Lint foi capaz de detectar um total de quatro categorias de vulnerabilidades do OWASP Mobile Top 10, sendo elas M1, M2, M5 e M6. Dentre essas categorias, a M1 (Uso Indevido da Plataforma) mostra-se a com maior ocorrência, correspondendo a mais da metade das vulnerabilidades detectadas. Por outro lado, conforme apresentado pelo gráfico, a vulnerabilidade M2 é a de menor ocorrência, representando 3,6% das vulnerabilidades encontradas pela ferramenta .

Também é importante destacar que o Android Lint foi a única ferramenta a identificar uma vulnerabilidade da categoria M6, a qual está relacionada à autorização insegura. De acordo com a Figura 22, essa categoria corresponde a $\frac{1}{4}$ das vulnerabilidades encontradas pelo Android Lint.

Figura 23 – Porcentagem de ocorrências das vulnerabilidades do OWASP Mobile Top 10 no FindSecBugs

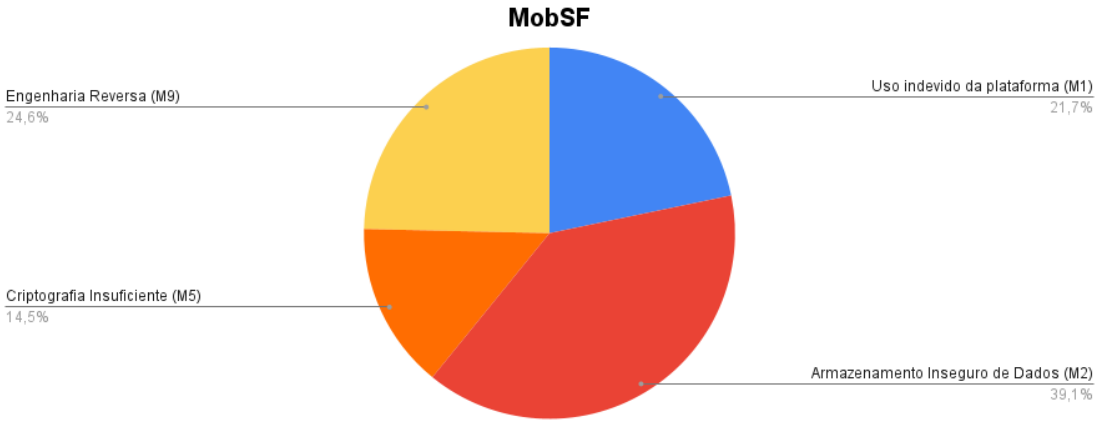


Fonte: Elaboração Própria

O FindSecBugs foi a ferramenta capaz de detectar a maior variedade de vulnerabilidades. Enquanto as outras ferramentas foram capazes de detectar apenas quatro tipos diferentes, o FindSecBugs detectou seis. Dentre essas vulnerabilidades, dois tipos não foram identificados por nenhuma das outras ferramentas. São eles M3 e M7, que tratam, respectivamente, de comunicação insegura e qualidade do código do cliente. Apesar disso, é importante destacar que não conseguiu identificar a M6 (Autorização Insegura), que foi detectada exclusivamente pelo Android Lint. As vulnerabilidades M4, M8 e M10 não foram identificadas, assim como ocorreu com as demais ferramentas.

Observa-se também uma boa capacidade dessa ferramenta na detecção de vulnerabilidades relacionadas ao armazenamento de dados (M2) e criptografia (M5). Dentre os tipos que foram identificados, os dois com menor índice foram M1 (Uso Indevido da Plataforma) e M3 (Comunicação Insegura), ambos com 2,2%.

Figura 24 – Porcentagem de ocorrências das vulnerabilidades do OWASP Mobile Top 10 no MobSF

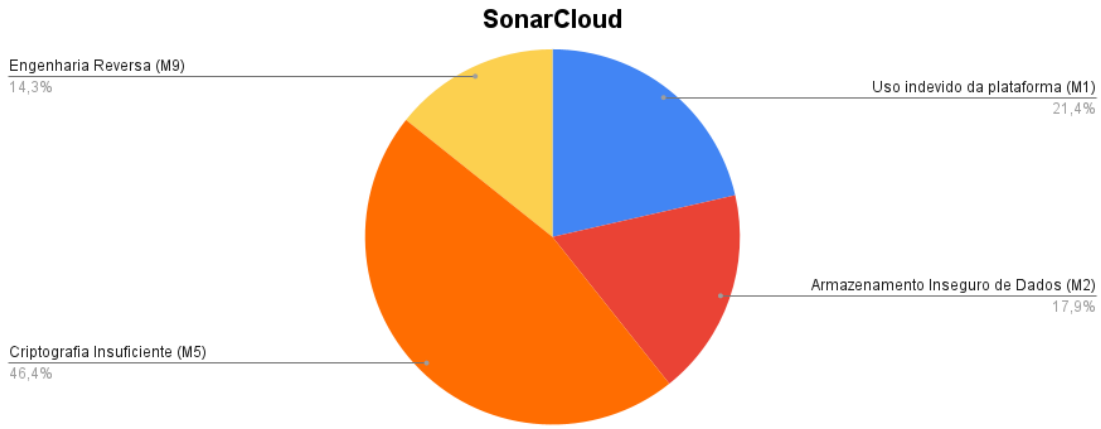


Fonte: Elaboração Própria

Com a análise das aplicações da amostra, o MobSF demonstrou a capacidade de identificar quatro categorias diferentes de vulnerabilidades, sendo elas M1, M2, M5 e M9. A maior parte das vulnerabilidades identificadas estão relacionadas ao armazenamento inseguro dos dados (M2), correspondendo a 39,1% do total detectado com a realização da análise estática.

No que diz respeito a menor ocorrência, destaca-se a categoria M5 (Criptografia Insuficiente), correspondendo a 14,5% das vulnerabilidades detectadas. Ainda assim, é importante destacar que o MobSF apresentou o maior equilíbrio no que diz respeito a quantidade de ocorrências de vulnerabilidades por categoria, apresentando 24,6% de diferença entre as categorias com mais e com menos vulnerabilidades detectadas.

Figura 25 – Porcentagem de ocorrências das vulnerabilidades do OWASP Mobile Top 10 no Sonar

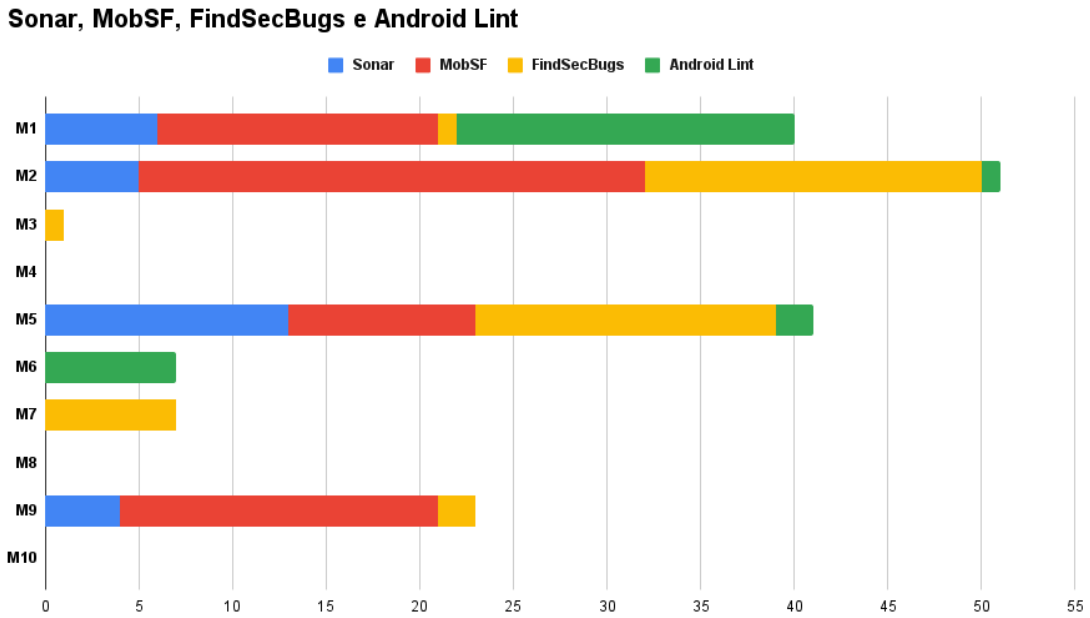


Fonte: Elaboração Própria

O SonarCloud foi capaz de identificar as mesmas quatro categorias de vulnerabilidade que o MobSF. No entanto, difere-se na porcentagem em que cada uma ocorre. Como apresentado na Figura 25, a maior parte das vulnerabilidades presentes na aplicação que foram detectadas pelo SonarCloud enquadram-se na categoria M5, ou seja, tratam-se de vulnerabilidades relacionadas à criptografia insuficiente. Por outro lado, a categoria com menos vulnerabilidades identificadas pelo SonarCloud corresponde a 14,3% das vulnerabilidades encontradas. Essa categoria diz respeito a engenharia reversa (M9).

A Figura 26 apresenta a quantidade total de ocorrências de cada categoria. Esse resultado é obtido pelo somatório da quantidade de ocorrências das categorias da lista do OWASP Mobile Top 10 encontradas por cada uma das ferramentas. Além disso, é possível diferenciar, conforme a legenda, qual a contribuição de cada ferramenta na identificação das categorias de vulnerabilidade. Dessa forma, o gráfico permite a compreensão do grau de dificuldade das ferramentas para identificar as diferentes categorias e fazer um comparativo da capacidade de detecção das ferramentas.

Figura 26 – Total de ocorrências das vulnerabilidades do OWASP Mobile Top 10



Fonte: Elaboração Própria

De acordo com a Figura 26, para a amostra selecionada, as ferramentas não foram capazes de detectar vulnerabilidades da categoria M4, M8 ou M10. Isso pode ser justificado pela inexistência de vulnerabilidade correspondente a essas categorias ou pela complexidade de se detectá-las.

As vulnerabilidades que se enquadram nas categorias M3, M6 ou M7 foram detectadas por no máximo uma dentre as quatro ferramentas selecionadas. No que diz respeito a M3 (Comunicação Insegura), apenas uma vulnerabilidade foi encontrada entre todas as ferramentas para todas as aplicações da amostra. Por outro lado, as categorias M6 e M7 tiveram sete vulnerabilidades relacionadas a cada uma delas, o que sugere uma dificuldade ainda maior quanto à detecção de vulnerabilidades dessas categorias por parte das ferramentas que não as identificaram. No caso da M6, as ferramentas que apresentaram maior dificuldade foram: SonarCloud, MobSF e FindSecBugs. Já, em relação a categoria M7, foram: SonarCloud, MobSF e Android Lint.

É possível observar no gráfico que a categoria com a maior quantidade de detecções de vulnerabilidades é a M2, seguida pela M5 e M1. De uma forma geral, essas categorias se mostraram as de maior facilidade de detecção para as ferramentas.

Embora todas as outras ferramentas tenham identificado uma grande quantidade de vulnerabilidades da categoria M2, o Android Lint foi capaz de identificar apenas uma, correspondendo apenas a 3,6% de suas vulnerabilidades detectadas, conforme observado na Figura 22. Dessa forma, é possível entender que o Android Lint apresentou dificuldades para detecção de vulnerabilidades dessa categoria.

No que diz respeito à categoria M1, embora todas as outras ferramentas tenham identificado uma boa quantidade de vulnerabilidades, o FindSecBugs detectou apenas uma. Dessa forma, entende-se que essa ferramenta apresenta uma maior dificuldade para detecção de vulnerabilidades relacionadas a categoria de uso indevido da plataforma.

É possível observar na Figura 26 que o o MobSF foi a ferramenta com a maior facilidade na detecção de vulnerabilidades das categorias M2 e M9.

No que diz respeito ao SonarCloud, observa-se que a ferramenta mostrou facilidade na detecção de vulnerabilidades da categoria M5. No entanto, diferente das outras ferramentas, não há nenhuma categoria na qual o SonarCloud se destaque como a ferramenta com maior facilidade de detecção.

As Tabelas 13, 14, 15, 16 e 17 mostram o desempenho de cada ferramenta no que diz respeito às categorias de vulnerabilidades identificadas em relação às aplicações Aegis, Amaze File Manager, OmniNotes, Retro Music Player e Wikipédia, respectivamente.

Tabela 13 – Resumo de identificação na aplicação Aegis

	Android Lint	**FindSecBugs**	**MobSF**	**Sonar**
M1			X	X
M2			X	X
M3				
M4				
M5		X	X	X
M6				
M7				
M8				
M9		X	X	
M10				

Na aplicação Aegis foram identificadas quatro das dez categorias de vulnerabilidades. A ferramenta que detectou mais tipos diferentes nessa aplicação foi o MobSF, seguido pelo SonarCloud, com três tipos detectados e pelo FindSecBugs com duas categorias. Além disso, o MobSF detectou todas as categorias que foram identificadas pelas demais. O Android Lint obteve o pior desempenho nesse caso, uma vez que não detectou nenhuma vulnerabilidade.

Tabela 14 – Resumo de identificação na aplicação Amaze File Manager

	Android Lint	**FindSecBugs**	**MobSF**	**Sonar**
M1	X	X	X	X
M2		X	X	X
M3				
M4				
M5		X	X	X
M6				
M7		X		
M8				
M9		X	X	X
M10				

Na aplicação Amaze File Manager a ferramenta que obteve melhor desempenho foi o FindSecBugs, que identificou cinco tipos diferentes de vulnerabilidades, sendo a que mais conseguiu identificar diferentes tipos dentro de uma mesma aplicação. As ferramentas MobSF e SonarCloud identificaram as mesmas categorias que o FindSecBugs, exceto a M7, que diz respeito à qualidade do código do cliente. Novamente, o Android Lint obteve o pior desempenho, detectando apenas uma categoria, também identificada pelas demais.

Tabela 15 – Resumo de identificação na aplicação Omni Notes

	Android Lint	**FindSecBugs**	**MobSF**	**Sonar**
M1			X	X
M2		X	X	X
M3				
M4				
M5	X		X	X
M6				
M7				
M8				
M9			X	
M10				

Na aplicação Omni Notes o MobSF obteve novamente o melhor desempenho, tendo detectado quatro categorias diferentes de vulnerabilidades. O SonarCloud performou de forma similar, mas não detectou a M9, sobre engenharia reversa. Em último lugar, o Android Lint e o FindSecBugs detectaram um tipo cada, respectivamente M2 (Armazenamento Inseguro de Dados) e M5 (Criptografia Insuficiente), ambas detectadas pelas duas primeiras.

Tabela 16 – Resumo de identificação na aplicação Retro Music Player

	Android Lint	**FindSecBugs**	**MobSF**	**Sonar**
M1	X		X	X
M2	X			X
M3		X		
M4				
M5				
M6				
M7				
M8				
M9			X	
M10				

Na aplicação Retro Music Player todas as aplicações detectaram dois tipos diferentes, exceto o FindSecBugs. O Android Lint e o SonarCloud detectaram as mesmas (M1 e M2), enquanto o MobSF detectou M1 e M9, mas não a M2. Sendo assim, vulnerabilidades de engenharia reversa foram detectadas exclusivamente pelo MobSF. O FindSecBugs detectou apenas um tipo, M3, que não foi detectado por mais nenhuma.

Tabela 17 – Resumo de identificação na aplicação Wikipédia

	Android Lint	**FindSecBugs**	**MobSF**	**Sonar**
M1	X		X	
M2				
M3				
M4				
M5				
M6	X			
M7				
M8				
M9			X	
M10				

Na aplicação Wikipédia ocorreu a menor variedade de categorias identificadas, com apenas três tipos diferentes. Duas ferramentas não detectaram nenhum tipo, o FindSecBugs e o SonarCloud. O Android Lint e o MobSF detectaram duas cada. Uma delas foi detectada pelas duas, a M1. O Android Lint identificou a M6 e o MobSF a M9.

Na detecção de vulnerabilidades com relação às diferentes aplicações, o MobSF se saiu melhor em quatro das cinco aplicações, tendo assim o melhor desempenho em comparação com as outras ferramentas. O FindSecBugs se saiu pior em três das cinco aplicações, demonstrando uma grande disparidade especificamente no Amaze File Manager, onde obteve o melhor desempenho de todas as ferramentas. O Android Lint também obteve a pior performance em três das cinco aplicações. Seu principal destaque foi na aplicação

Wikipédia, cenário em que apenas duas ferramentas detectaram alguma vulnerabilidade. O SonarCloud obteve melhor desempenho apenas no Retro Music Player, porém obteve o pior desempenho apenas uma vez, na aplicação Wikipédia.

Assim, é possível concluir que o MobSF foi capaz de identificar a maior variedade de vulnerabilidades nessa comparação de aplicação por aplicação, não obtendo o melhor desempenho apenas em uma situação, na aplicação Amaze File Manager, onde ficou em segundo lugar, ao lado do SonarCloud e atrás do FindSecBugs.

8.3 Relato dos Pesquisadores

O terceiro critério estabelecido na pesquisa para realizar o comparativo entre as ferramentas de análise estática mais acessíveis aos desenvolvedores, conforme a Q1, trata-se da experiência proporcionada aos pesquisadores. O Capítulo 4 apresenta os relatos iniciais dos pesquisadores com respeito a cada uma das ferramentas. No entanto, após as execuções das análises estáticas na amostra utilizando cada uma das ferramentas selecionadas, é possível apresentar um relato de experiência mais completo e fazer comparações das experiências dos pesquisadores com respeito ao uso das ferramentas.

Primeiramente, o tempo despendido para realizar a análise estática do código é destacado como uma desvantagem do SonarCloud em relação às outras ferramentas. Conforme observado na Tabela 6 apresentada no capítulo 5, enquanto as demais ferramentas apresentavam o relatório de análise em tempo inferior a 20 segundos, o SonarCloud apresentava em mais de dois minutos. No entanto, é válido destacar que, para três das cinco aplicações da amostra, o Android Lint também consumiu tempo superior a um minuto para apresentar resultados, chegando até três minutos e 25 segundos. A tabela também mostra que as ferramentas MobSF e FindSecBugs realizam a atividade de análise estática de código em um tempo consideravelmente menor em relação às outras ferramentas.

O fator tempo de análise pode ser um fator mais agravante para o Android Lint devido ao fato de que, por ser integrado ao Android Studio, a ferramenta não salva as análises realizadas, como acontece com o MobSF e o SonarCloud. Sendo assim, toda vez que for necessário realizar a análise, será despendido, em média, o mesmo tempo. O mesmo acontece com o FindSecBugs, porém, o tempo apresentado pela ferramenta é consideravelmente menor. No entanto, é importante destacar a funcionalidade de exportação do relatório de análise presente nas duas ferramentas. No caso do Android Lint, o relatório exportado não apresenta alguns recursos importantes, tais como informações adicionais a respeito das vulnerabilidades encontradas.

Assim como o Android Lint, o FindSecBugs está integrado no Android Studio. Por conta disso, essas ferramentas se mostraram de fácil utilização. Além disso, por permitir a configuração de nível de esforço na detecção, nível de confiança e categorias das

vulnerabilidades, o FindSecBugs se apresenta como uma ferramenta que dá mais liberdade para o desenvolvedor. O Android Lint também apresenta uma boa diversidade de configurações, incluindo categorias desejadas, níveis de severidade e exclusão de códigos de teste da aplicação.

Um recurso importante apresentado pelo SonarCloud é a capacidade de automatizar a execução da análise por meio da criação de uma *pipeline* no GitHub. Porém, essa atividade exige um esforço maior para realizar a configuração, sendo um grau de dificuldade maior em relação às configurações necessárias para executar as análises manualmente a partir das demais ferramentas ou, até mesmo, em relação à execução manual do próprio SonarCloud. Ainda assim, trata-se de um recurso útil apresentado pela ferramenta.

O MobSF possui uma instalação relativamente complexa, pois demanda a instalação de diversas dependências. Há, no entanto, uma versão para utilizar com Docker, o que facilita o processo. Uma vez instalado, o MobSF mostrou-se de fácil utilização, visto que é necessário apenas iniciar a ferramenta e fazer upload do código fonte em uma pasta compactada ou do pacote APK da aplicação. Com respeito ao relatório apresentado pela ferramenta após a execução da análise estática, percebe-se um relatório completo. No entanto, se o foco estiver apenas nas vulnerabilidades apresentadas no código, a grande quantidade de informações apresentadas no relatório pode ser uma desvantagem. Para contornar esse problema, existe o recurso de navegação pelas seções do relatório. Além disso, há um problema relacionado com os arquivos Kotlin, em comparação com Java, pois no caso do primeiro a ferramenta não mostra o conteúdo do arquivo onde foram indicadas as vulnerabilidades. Há também possibilidade de gerar um relatório bem estruturado no formato pdf, porém não inclui as linhas em que foram encontradas falhas dentro de um mesmo arquivo do projeto. Assim, não é possível identificar quantos problemas foram encontrados em um mesmo arquivo.

As impressões iniciais sobre o relatório apresentado pelo SonarCloud foram mantidas, visto que se trata de um relatório de fácil compreensão e que apresenta recursos úteis aos desenvolvedores, incluindo, para cada uma das vulnerabilidades listadas pela ferramenta, um visualizador de arquivo indicando o local da vulnerabilidade, a justificativa para ser uma vulnerabilidade, como corrigir a vulnerabilidade e o mapeamento para o CWE e os projetos do OWASP, tais como Mobile Top 10 e Top 10.

Em relação ao relatório apresentado pelo Android Lint, trata-se do mais simples em comparação com os relatórios gerados pelas demais ferramentas. Nela, só se pode visualizar a descrição da vulnerabilidade pela própria IDE. Também é importante destacar que, mesmo após realizar a configuração para apresentar apenas vulnerabilidades da categoria de segurança, é possível que o relatório apresente vulnerabilidades de outras categorias. Assim, foi necessário realizar uma breve análise para desconsiderar essas vulnerabilidades de categorias não selecionadas na configuração. Além disso, algumas das

configurações escolhidas não persistem totalmente para as próximas análises, tornando necessário configurar novamente em futuras execuções.

Com base na experiência de cada pesquisador, as Tabelas 18 e 19 apresentam, para cada ferramenta, uma pontuação de 1 a 5 dada pelos pesquisadores Kleidson Alves e Lucas Rodrigues, respectivamente, referente aos seguintes aspectos: facilidade de uso da ferramenta, facilidade de configuração e qualidade do relatório apresentado, sendo 5 o valor mais positivo dado pelo pesquisador de acordo com a sua experiência e 1 o valor mais negativo.

Tabela 18 – Avaliação das ferramentas por Kleidson Alves

	Facilidade de uso	**Configuração**	**Relatório**
Android Lint	3	4	2
FindSecBugs	4	4	4
MobSF	4	5	4
Sonar	4	3	5

Tabela 19 – Avaliação das ferramentas por Lucas Rodrigues

	Facilidade de uso	**Configuração**	**Relatório**
Android Lint	4	3	2
FindSecBugs	5	5	4
MobSF	3	5	4
Sonar	4	5	5

No critério de facilidade de uso, foram avaliados aspectos como a facilidade de aprendizado no uso das funcionalidades e a interface da ferramenta. Para o critério de configuração, foram consideradas as opções fornecidas para customizar a análise e a geração dos relatórios, como a sensibilidade da detecção e as categorias desejadas. Para o critério de relatório foram levados em conta questões como a organização do relatório, facilidade de compreensão e disponibilização de informações fornecidas a respeito das falhas, como o mapeamento, exemplos em código da vulnerabilidade em questão e a descrição de cada uma.

Com as pontuações feitas pelos pesquisadores, justificadas pela experiência na realização deste estudo, torna-se possível encontrar uma pontuação média para os critérios de facilidade de uso, facilidade de configuração e qualidade do relatório analisados pelos pesquisadores em relação às ferramentas selecionadas. Nesse sentido, a Tabela 20 apresenta, para cada ferramenta, as médias da pontuação para os critérios analisados.

Tabela 20 – Média da avaliação dos pesquisadores

	Facilidade de uso	**Configuração**	**Relatório**
Android Lint	3,5	3,5	2
FindSecBugs	4,5	4,5	4
MobSF	3,5	5	4
Sonar	4	4	5

8.4 Vulnerabilidades Mais Comuns

Com o objetivo de responder a Q2 e identificar as vulnerabilidades com mais ocorrências, as vulnerabilidades detectadas por todas as ferramentas foram relacionadas, totalizadas e ordenadas na Tabela 21. Assim, torna-se possível observar as ocorrências mais comuns de acordo com o mapeamento que foi realizado dentre as vulnerabilidades listadas pelo OWASP Mobile Top 10. É importante notar que foram descontadas as vulnerabilidades iguais detectadas por diferentes ferramentas, uma vez que o foco desta análise é na identificação das vulnerabilidades mais comuns no contexto de aplicações móveis, e não na ferramenta.

Tabela 21 – Ranking das categorias com mais ocorrências

Vulnerabilidade	**Descrição**	**Quantidade**
M2	Armazenamento Inseguro de Dados	51
M1	Uso Indevido da Plataforma	40
M5	Criptografia Insuficiente	33
M9	Engenharia Reversa	23
M6	Autorização Insegura	7
M7	Qualidade do Código do Cliente	7
M3	Comunicação Insegura	1
M4	Autenticação Insegura	0
M8	Adulteração de Código	0
M10	Funcionalidade Irrelevante	0

Pode-se observar que a vulnerabilidade mais comum foi a M2, referente ao armazenamento inseguro de dados, indicando que as ferramentas em geral encontram maior facilidade na detecção desse tipo de falha no código. Além disso, indica que problemas de segurança no armazenamento e manipulação de dados se fazem bastante presentes nas aplicações para dispositivos móveis, que ficam assim suscetíveis a fraudes e aquisição de dados sensíveis dos usuários por parte de ataques maliciosos.

Dentre as vulnerabilidades que não foram detectadas por nenhuma das ferramentas, pode-se citar a M4 (Autenticação Insegura), M8 (Adulteração de Código) e M10 (Funcionalidade Irrelevante), o que poderia indicar que as ferramentas de teste estático

encontram maiores dificuldades na análise de falhas de segurança relacionadas a essas situações.

Com relação a vulnerabilidade M10, isso pode ser explicado pela dificuldade que uma ferramenta de teste estático teria para compreender que uma determinada funcionalidade não deveria fazer parte da aplicação. Um exemplo seria uma funcionalidade utilizada para realizar testes em ambiente de desenvolvimento e que não deveria fazer parte do código em produção. Por outro lado, é possível que tenham perdido relevância ou ainda que, em razão da similaridade que pode ser estabelecida dentre algumas das vulnerabilidades, ela seja preterida pela outra no momento da classificação e do mapeamento. Isso pode ser ser notado pelo fato de que na lista mais recente do OWASP Mobile Top 10 de 2023, ainda em fase de desenvolvimento, algumas dessas vulnerabilidades foram mescladas. A M4, por exemplo, apresenta diversas similaridades com a M6 (Autorização Insegura), enquanto a M8 pode ser relacionada com a M7 (Qualidade do Código do Cliente), as quais tem sete vulnerabilidades detectadas cada uma nos resultados obtidos.

9 Considerações Finais

Durante a realização dessa pesquisa, foi possível compreender como é possível realizar um comparativo entre as ferramentas de análise estática mais acessíveis aos desenvolvedores. Além disso, foi possível identificar quais as vulnerabilidades mais comuns presentes em aplicações móveis para Android.

O comparativo entre as ferramentas pode ser feito levando em consideração as principais características apresentadas por elas, a experiência dos pesquisadores com o uso das ferramentas e o desempenho apresentado por elas quanto a capacidade de detecção de vulnerabilidades. Assim, tornou-se possível compreender em quais situações cada ferramenta se sobressai em comparação com as demais.

Para a realização da comparação se fez necessária a realização de diversas etapas anteriores. Primeiramente, a execução das análises, combinando cada ferramenta com cada aplicação. Essa etapa apresentou desafios, como a preparação do ambiente mais complexa no caso do MobSF e SonarCloud, enquanto nas outras foi mais simples devido a integração na própria IDE. A etapa de análise de falsos positivos exigiu aprofundamentos nos conhecimentos de segurança e análise do código da aplicação, que por muitas vezes é complexo, a fim de determinar se seria ou não falso positivo. Além disso, a etapa de mapeamento demandou pesquisas no catálogo de vulnerabilidades, a fim de relacionar as vulnerabilidades detectadas pelas ferramentas com as que estão no CWE e na lista do OWASP.

De acordo com os resultados apresentados pela pesquisa, é possível observar que as ferramentas levaram vantagem sobre outras em diferentes aspectos. Por exemplo, o MobSF se destacou em comparação às demais ferramentas em relação a detecção de vulnerabilidades. No entanto, essa mesma ferramenta apresentou a segunda pior precisão em sua análise. Por outro lado, a aplicação com menos detecção de verdadeiros positivos, o Android Lint, apresentou a menor quantidade de falsos positivos e a maior precisão. Isso demonstra que a ferramenta com maior precisão não necessariamente será a melhor na detecção de vulnerabilidades, visto que algumas importantes vulnerabilidades passaram despercebidas pelo Android Lint.

Os resultados também mostraram que o FindSecBugs, em relação a toda amostra de aplicações, foi a ferramenta capaz de identificar mais categorias diferentes de vulnerabilidades. No entanto, na comparação aplicação por aplicação, o MobSF foi a ferramenta que se mostrou superior às demais, por ter detectado maior quantidade de tipos de vulnerabilidades mais vezes em cada cenário.

O SonarCloud se mostrou muito preciso na identificação de vulnerabilidades reais,

devido a sua capacidade de separar as vulnerabilidades dentre aquelas que se tem mais confiança e aquelas que há uma maior grau de incerteza, podendo ou não se tratar de falhas de segurança.

Orientada pelas categorias apresentadas pela lista definitiva do projeto OWASP Mobile Top 10, a pesquisa foi capaz de mostrar que as vulnerabilidades mais comuns estão relacionadas com o armazenamento inseguro de dados e com o uso indevido da plataforma, listadas pelo OWASP como M2 e M1, respectivamente. Além disso, não foram encontradas vulnerabilidades relacionadas à autenticação insegura, adulteração de código e funcionalidade irrelevante. Isso pode indicar a incapacidade das ferramentas estudadas de detectar vulnerabilidades dessa categoria ou a inexistência dessas vulnerabilidades nas aplicações, mostrando-se as menos comuns.

Para os trabalhos futuros, recomenda-se aumentar a amostra de aplicações, levando em consideração a seleção de aplicações de diferentes categorias. Também há a possibilidade de aumentar a quantidade de ferramentas de teste estático, a fim de comparar com diferentes ferramentas e com diferentes aplicações. Assim, seria possível observar se os resultados obtidos seguiriam a mesma direção observada neste estudo. Além disso, poderiam ser adicionadas diferentes perspectivas de comparação, como a facilidade de configurar uma *pipeline* em ferramentas de integração contínua.

Outra possibilidade é realizar o estudo com a lista de vulnerabilidades mais recente do OWASP para aplicações móveis, a qual está sendo desenvolvida. Assim, a comparação poderia ser atualizada conforme os tipos de vulnerabilidade que tem sido mais explorados desde a realização da última lista definitiva, finalizada no ano de 2016.

Uma terceira possibilidade para um trabalho futuro é a inclusão dos *hotspots* do SonarCloud na lista de vulnerabilidades detectadas pela ferramenta, além de uma configuração diferente das configurações padrão para as demais ferramentas.

Referências

ALANDA, A. et al. Mobile Application Security Penetration Testing Based on OWASP. *IOP Conference Series: Materials Science and Engineering*, v. 846, n. 1, p. 012036, maio 2020. ISSN 1757-899X. Publisher: IOP Publishing. Disponível em: <https://dx.doi.org/10.1088/1757-899X/846/1/012036>. Citado 3 vezes nas páginas 12, 22 e 23.

ANTUNES, N.; VIEIRA, M. On the Metrics for Benchmarking Vulnerability Detection Tools. In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. [s.n.], 2015. p. 505–516. ISSN: 2158-3927. Disponível em: <https://ieeexplore.ieee.org/document/7266877>. Citado na página 60.

AU, M. H.; CHOO, R. *Mobile Security and Privacy: Advances, Challenges and Future Research Directions*. Cambridge: Syngress, 2016. ISBN 978-0-12-804746-0. Citado na página 21.

AUTILI, M. et al. Software engineering techniques for statically analyzing mobile apps: research trends, characteristics, and potential for industrial adoption. *Journal of Internet Services and Applications*, v. 12, n. 1, p. 3, jul. 2021. ISSN 1869-0238. Disponível em: <https://doi.org/10.1186/s13174-021-00134-x>. Citado 3 vezes nas páginas 15, 16 e 24.

AVANCINI, A.; CECCATO, M. Security testing of the communication among Android applications. In: *2013 8th International Workshop on Automation of Software Test (AST)*. [S.l.: s.n.], 2013. p. 57–63. Citado na página 20.

BASUTAKARA, A. S. A Review of Static Code Analysis Methods for Detecting Security Flaws. v. 23, n. 6, 2021. Citado na página 24.

BORJA, T. et al. Risk Analysis and Android Application Penetration Testing Based on OWASP 2016. In: ROCHA, et al. (Ed.). *Information Technology and Systems*. Cham: Springer International Publishing, 2021. (Advances in Intelligent Systems and Computing), p. 461–478. ISBN 978-3-030-68285-9. Citado 2 vezes nas páginas 22 e 23.

BRERETON, P. et al. Using a Protocol Template for Case Study Planning. In: . [s.n.], 2008. Disponível em: <https://scienceopen.com/hosted-document?doi=10.14236/ewic/EASE2008.5>. Citado na página 26.

CHELL, D. et al. *The Mobile Application Hacker's Handbook*. New York: John Wiley & Sons, Incorporated, 2015. ISBN 978-1-118-95851-3. Citado 3 vezes nas páginas 20, 21 e 22.

DWIVEDI, H.; CLARK, C.; THIEL, D. *Mobile Application Security*. [S.l.]: Mcgraw-hill, 2010. ISBN 978-0-07-163356-7. Citado 2 vezes nas páginas 20 e 21.

ELDER, S. et al. Do I really need all this work to find vulnerabilities?: An empirical case study comparing vulnerability detection techniques on a Java application. *Empirical Software Engineering*, v. 27, n. 6, p. 154, nov. 2022. ISSN 1382-3256, 1573-7616. Disponível em: <https://link.springer.com/10.1007/s10664-022-10179-6>. Citado 2 vezes nas páginas 20 e 24.

FAHL, S. Web & Mobile Security Knowledge Area. *Web & Mobile Security Knowledge Area*, 2021. Disponível em: <https://www.cybok.org/media/downloads/Web_Mobile_Security_v1.0.1.pdf>. Citado 3 vezes nas páginas 11, 17 e 20.

GANAPATI, S. Using Mobile Apps in Government. 2015. Citado na página 11.

ISLAM, R.; ISLAM, R.; MAZUMDER, T. A. Mobile Application and Its Global Impact. *International Journal of Engineering*, v. 10, n. 06, 2010. Citado 2 vezes nas páginas 11 e 15.

KULKARNI, K.; JAVAID, A. Y. Open Source Android Vulnerability Detection Tools: A Survey. 2018. Citado na página 24.

LEE, W. van der; VERWER, S. Vulnerability Detection on Mobile Applications Using State Machine Inference. In: *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. [S.l.: s.n.], 2018. p. 1–10. Citado na página 22.

MARTIN, A. et al. Introduction to CyBOK Knowledge Area. *Introduction to CyBOK Knowledge Area*, 2021. Disponível em: <https://www.cybok.org/media/downloads/Introduction_v1.1.0.pdf>. Citado na página 19.

MUCCINI, H.; FRANCESCO, A. D.; ESPOSITO, P. Software testing of mobile applications: Challenges and future research directions. In: *2012 7th International Workshop on Automation of Software Test (AST)*. Zurich: IEEE, 2012. p. 29–35. ISBN 978-1-4673-1822-8 978-1-4673-1821-1. Disponível em: <http://ieeexplore.ieee.org/document/6228987/>. Citado 3 vezes nas páginas 11, 12 e 15.

NUNES, P. et al. Benchmarking Static Analysis Tools for Web Security. *IEEE Transactions on Reliability*, v. 67, n. 3, p. 1159–1175, set. 2018. ISSN 0018-9529, 1558-1721. Disponível em: <https://ieeexplore.ieee.org/document/8399530/>. Citado na página 12.

OWASP. *OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation*. 2023. Disponível em: <https://owasp.org/>. Citado 3 vezes nas páginas 19, 22 e 23.

OYETOYAN, T. D.; CHAIM, M. L. Comparing Capability of Static Analysis Tools to Detect Security Weaknesses in Mobile Applications. 2017. Citado na página 24.

PAN, Y. et al. A Systematic Literature Review of Android Malware Detection Using Static Analysis. *IEEE Access*, v. 8, p. 116363–116379, 2020. ISSN 2169-3536. Conference Name: IEEE Access. Citado na página 24.

PAYER, M. Software Security. jul. 2021. Citado 3 vezes nas páginas 17, 18 e 19.

PRAHOFER, H. et al. *Opportunities and challenges of static code analysis of IEC 61131-3 programs*. [S.l.: s.n.], 2012. Journal Abbreviation: IEEE International Conference on Emerging Technologies and Factory Automation, ETFA Pages: 8 Publication Title: IEEE International Conference on Emerging Technologies and Factory Automation, ETFA. ISBN 978-1-4673-4737-2. Citado na página 12.

QIAN, K.; PARIZI, R. M.; LO, D. OWASP Risk Analysis Driven Security Requirements Specification for Secure Android Mobile Software Development. In: *2018 IEEE Conference on Dependable and Secure Computing (DSC)*. [S.l.: s.n.], 2018. p. 1–2. Citado na página 22.

RANSOME, J.; MISRA, A. *Core Software Security: Security at the Source*. New York: Auerbach Publishers, Incorporated, 2013. ISBN 978-1-4665-6096-3. Citado 4 vezes nas páginas 17, 18, 19 e 23.

SCHLEIER, S. et al. *OWASP MASVS - OWASP Mobile Application Security*. 2023. Disponível em: <https://mas.owasp.org/MASVS/>. Citado na página 20.

SCHLEIER, S. et al. *OWASP MASTG - OWASP Mobile Application Security*. 2022. Disponível em: <https://mas.owasp.org/MASTG/>. Citado 4 vezes nas páginas 16, 17, 20 e 24.

SCHMEELK, S.; TAO, L. A Case Study of Mobile Health Applications: The OWASP Risk of Insufficient Cryptography. *Journal of Computer Science Research*, v. 4, n. 1, p. 22–31, fev. 2022. ISSN 2630-5151. Disponível em: <https://journals.bilpubgroup.com/index.php/jcsr/article/view/4271>. Citado 2 vezes nas páginas 21 e 22.

SHIREY, R. *Internet Security Glossary*. [S.l.], 2000. Num Pages: 212. Disponível em: <https://datatracker.ietf.org/doc/rfc2828>. Citado na página 20.

VALDIVIA, G. *5 Pilares de Segurança da Informação nas Empresas*. 2022. Disponível em: <https://www.gatinfosec.com/blog/5-pilares-da-seguranca-da-informacao/>. Citado na página 18.

VELU, V. K. *Mobile Application Penetration Testing*. Birmingham: Packt Publishing Ltd, 2016. ISBN 978-1-78588-869-4. Citado 4 vezes nas páginas 11, 12, 15 e 16.

WASSERMAN, A. I. Software engineering issues for mobile application development. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research*. New York: Association for Computing Machinery, 2010. (FoSER '10), p. 397–400. ISBN 978-1-4503-0427-6. Disponível em: <https://doi.org/10.1145/1882362.1882443>. Citado na página 12.

Apêndices

APÊNDICE A – Repositório de Relatórios e Planilhas

O repositório contendo os relatórios gerados pelas ferramentas Android Lint, FindSecBugs e MobSF, além das planilhas para a contagem de falsos positivos e mapeamento completo das vulnerabilidades pode ser acessado pelo seguinte link: repositório.

A organização com os resultados das análises pelo SonarCloud pode ser acessada pelo seguinte link: organização.