INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Cálculo Automático do Nível de Risco e Ameaça no Desenvolvimento de Aplicações Android

Nuno Alexandre Realista Ferreira

Mestrado em Engenharia de Telecomunicações e Informática

Orientador:
Professor Doutor Carlos José Corredoura Serrão, Professor Associado
ISCTE-Instituto Universitário de Lisboa

Outubro, 2020

Agradecimento

Um enorme agradecimento aos meus pais, pois eles foram a minha maior motivação para prosseguir com a minha vida académica.

Quero também agradecer ao meu orientador Carlos Serrão, por estar sempre disponível para me auxiliar quando precisei.

Um agradecimento a todos os meus amigos por me ajudarem a fazer este percurso com muito mais facilidade, eles sabem quem são.

Resumo

Utilizadores de dispositivos móveis disponibilizam cada vez mais os seus dados a aplicações posteriormente instaladas para fazerem um conjunto de operações importantes . É preciso ter em conta que esses dados muitas vezes são pessoais e sensíveis. As aplicações instaladas nesses dispositivos guardam localmente esses dados e são responsáveis pelo seu transporte na internet, deste modo, quem desenvolve essas aplicações tem a obrigação de proteger os dados. A segurança de uma aplicação deve ser logo implementada durante o desenvolvimento, os programadores devem ser educados relativamente a desenvolvimento seguro.

Neste trabalho é proposto um sistema que permite efetuar uma classificação automática de risco para aplicações Android. O sistema devolve uma métrica numérica que representa o nível de segurança de uma aplicação chegando a este valor com a ajuda de ferramentas de análise estática. Deste modo, programadores têm apoio à decisão, podem definir os seus próprios limites em relação ao valor de risco mínimo que querem cumprir antes de publicar uma aplicação nas respetivas lojas. O mesmo aplica-se às lojas de aplicações que podem usar este sistema para controlo de qualidade filtrando aplicações ao definirem um valor de risco máximo. O sistema explica os valores de risco obtidos para que programadores e gestores de lojas de aplicação possam compreender melhor o risco envolvido na aplicação. De modo a validar o sistema implementado, foram realizados testes com várias aplicações descarregadas em lojas assim como a aplicações propositadamente vulneráveis. Desta forma podendo comparar o índice de risco calculado para cada uma delas.

Palavras-chave: análise, classificação, vulnerabilidades, android, programadores, risco, ameaça, CVSS

Abstract

Mobile device users make available their data more often to make a set of important operations using third-party applications. And it needs to be evident that this data, most of the times is personal and sensitive. The third party applications installed on those devices, locally store that data and are responsible for their transport over the internet, accordingly, who develops this apps has the obligatoriness to pretect the data. The security of na application must be implemented during its development, programmers must be educated to develop with security awreness.

In this work we present an automated risk classification system for Android Apps. The system gives a numeric value that represents the security level of an app achieving this value with the help of static code analysis tools. With this system, developers have a support for decision, they can define their own limits relatively to the risk value they want to accomplish before publishing an app in the respective app stores. The same applies to app stores, they can use this system for quality control, filtering the apps by defining a maximum risk value. The system also allows developers and QA managers to understand how the risk was achieved to make them understand more the involved risk in the application. To validate the implemented system, it was performed a set of tests with apps downloaded from app stores as well as apps intentionally developed with vulnerabilities to compare the risk calculated between them.

Keywords: analysis, classification, vulnerabilities, android, developers, risk, threat, CVSS

Índice

Índice de Tabelas

Índice de Figuras

Lista de Acrónimos

APK – Android Package

CVE – Common Vulnerability Exposure

CVSS – Common Vulnerability Scoring System

OWASP – Open Web Application Security Project

NVD – National Vulnerability Database

API – Application Programing Interface

URI – Uniform Resource Identifier

IPC - Inter-Process Communication

RAM – Random Access Memory

SDK – Software Development Kit

JAR – Java ARchive

DEX – Dalvik Excutable

JVM – Java Virtual Machine

ADB – Android Debug Bridge

1 Introdução

Esta dissertação visa a contribuir para um dos bens comuns mais importantes para as pessoas, a segurança. Neste caso uma contribuição para a segurança no mundo digital e tendo em conta que vamos nos tornando cada vez mais virtuais e que a segurança é sempre um interesse de prioridade, neste mundo em crescimento este bem comum não deve ser posto de lado. Neste primeiro capítulo será abordado o problema em questão e qual a motivação para a sua resolução.

1.1 Motivação

Atualmente a utilização do *smartphone* é cada vez mais fundamental na vida de uma pessoa. Através de diversas aplicações, os utilizadores conseguem fazer vários procedimentos importantes para as suas vidas que implicam o uso dos seus dados pessoais, como por exemplo, transferências bancárias, marcar consultas médicas, fazer compras on-line, registar características biométricas e muitos mais. A dependência deste tipo de utilização tende a aumentar com o passar dos anos assim como o número de aplicações no mercado. Atualmente a quota de mercado Android a nível mundial ocupa os 75,85% (Statcounter, 2019). Sendo o mercado dominante, existe uma enorme preocupação perante o uso de aplicações para este sistema operativo. Apesar de haver um esforço enorme por parte da Google e de outras empresas para garantir a segurança das aplicações, nem sempre é possível detetar se estão seguras antes de chegar aos utilizadores. Muitas destas falhas devem-se á fraca implementação de práticas de segurança durante o desenvolvimento, os programadores têm forte conhecimento sobre implementação e desenho, mas têm falta de conhecimento sobre os riscos durante o mesmo procedimento. No mundo profissional, muitas vezes o desenvolvimento é focado no objetivo funcional e com tempo limitado de modo a cumprir as datas definidas para a conclusão de um projeto. Este foco não permite a programadores profissionais investirem num desenvolvimento cuidado pensado nas possíveis falhas de segurança, deixando esta importante matéria para depois, quando o projeto já está em produção e a circular com dados pessoais dos seus utilizadores.

Uma vulnerabilidade pode ser um *bug*, uma falha, ou um evento num sistema, dispositivo, ou serviço que pode causar implicitamente ou explicitamente uma falha de confidencialidade, integridade ou de disponibilidade (Khazaei et al., 2016). A decisão de publicar uma aplicação no mercado pode ser reforçada se o programador tiver algum conhecimento de quais vulnerabilidades podem estar presentes no final do seu trabalho, se estas são facilmente mitigadas e se souber o nível de severidade delas. Associar um nível de risco à aplicação desenvolvida reforça e ajuda muito mais nesta decisão.

A principal motivação para esta dissertação é de investigar e definir um mecanismo que calcule automaticamente o nível de risco de uma aplicação depois de esta estar desenvolvida ou ao longo do

desenvolvimento. A ideia é implementar um protótipo de cálculo num mecanismo que identifica vulnerabilidades em aplicações móveis. Deste modo, um programador pode ter noção do nível de segurança da aplicação que desenvolveu, pode saber onde precisa melhorar certos aspetos durante o desenvolvimento e as organizações podem implementar metas de nível de segurança e fazer uma melhor gestão de risco no desenvolvimento de aplicações.

1.2 Enquadramento

Estudos sobre a segurança de software têm sido realizados ao longo do tempo, vários tópicos têm sido discutidos por investigadores, a inclusão de protocolos de segurança e de padrões para uma construção segura de sistemas, testes de segurança de software, deteção de vulnerabilidades, previsão de ataques e sistemas de deteção de intrusão, só para nomear alguns exemplos. Isto mostra que desenvolver software sem falhas de segurança que originam vulnerabilidades, ainda é muito difícil, apesar do gigante avanço em técnicas de desenvolvimento de software (Alenezi & Almomani, 2018).

Vários estudos apontam para a importância da quantificação do nível de segurança para sistemas informáticos. O uso de uma medida universal pode ter grandes impactos na gestão de risco de um software. Para as aplicações móveis, neste caso para o mercado Android, é fundamental que se possa gerir o risco antes de serem lançadas para o mercado. Os autores de (Corral & Fronza, 2015) estudaram o impacto da qualidade do código no mercado, o autor de (Nancy R. Mead, Julia H. Allen, Sean Barnum, Robert J. Ellison, 2008), refere a importância das tomadas de decisão por parte de gestores de projetos consoante o nível de risco do software desenvolvido e que detetar falhas de segurança antes de um produto ser lançado, pode prevenir enormes perdas económicas para as organizações.

Métricas são uma medida quantificadora. Métricas de segurança são indicadores quantitativos para atributos de segurança de um sistema de informação ou tecnológico. Métricas ajudam-nos a entender qualidade e consistência. Métricas fornecem uma estrutura universal para trocar ideias, para medir a qualidade de um produto ou serviço e para melhorar o processo. Não se pode melhorar a segurança se não for possível medi-la (J. A. Wang et al., 2009). Um dos sistemas de medição mais utilizados atualmente por organizações, é o CVSS (*Common Vulnerability Scoring System*). O CVSS é uma ferramenta aberta para comunicação das características e severidade de uma vulnerabilidade de software (FIRST, 2019). Embora vários estudos apresentam que este sistema de medição ainda apresenta desvantagens e que não reflete a verdadeira situação do risco de uma vulnerabilidade, o vasto uso do CVSS por parte de organizações mostra que há uma confiança neste sistema (Petraityte et al., 2018). Os autores de (Petraityte et al., 2018) apresentam uma proposta de cálculo de risco para aplicações Android e iOS usando o CVSS, em (Khazaei et al., 2016) mostram resultados para a previsão de um *score* CVSS apenas analisando a descrição de vulnerabilidades com técnicas de *Text Mining*.

Atualmente existem vários métodos para calcular o nível de risco de uma aplicação Android, com acesso ao código-fonte como a previsão do risco recorrendo a métricas de código estático, (Alenezi & Almomani, 2018) e (Rahman et al., 2017), ou sem acesso ao código fonte, através da análise de vulnerabilidades, (Petrait yte et al., 2018) e (Khazaei et al., 2016). Embora ainda seja um problema a ser explorado, estudos mostram que é possível quantificar uma aplicação Android quanto ao seu nível de segurança.

A Aptoide juntamente com a unidade de investigação ISTAR-IUL do ISCTE-Instituto Universitário de Lisboa, estão a desenvolver um projeto com o objetivo de apoiar os programadores para questões de segurança durante o desenvolvimento das suas aplicações. Esta dissertação está incluída nesse projeto cujo nome é AppSentinel.

1.3 Questões de Investigação

Este trabalho procura responder à seguinte questão de investigação: Será possível calcular automaticamente uma métrica baseada nas vulnerabilidades de segurança de uma aplicação Android que permita determinar o nível de risco da mesma?

1.4 Objetivos

O que se pretende com esta dissertação é o desenvolvimento de um mecanismo de cálculo automático para avaliar o nível de risco de aplicações Android. Com o objetivo de orientar os programadores a olharem mais para a segurança como um fator muito importante durante o desenvolvimento.

Contudo, um dos grandes problemas na identificação de uma aplicação vulnerável é concluir se realmente pode ser uma ameaça grave ou não. O entendimento e a decisão do nível de severidade podem ser muitas vezes subjetivos se não houver nenhum sistema de classificação.

O objetivo do projeto é fazer uma análise aos APKs (android packages) que são submetidos, não tendo acesso ao código fonte. O sistema deve retornar uma análise das vulnerabilidades identificadas por várias ferramentas que recorrem a métodos de análise estática ou recorrendo a engenharia reversa para avaliar a implementação do sistema, consoante essas vulnerabilidades é associado um nível de risco para a aplicação e assim o programador pode facilmente identificar e resolver os problemas em questão. O nível de risco calculado é fundamental pois pode ser a base da decisão de publicar uma aplicação ou não.

1.5 Método de Investigação

O método de investigação usado no âmbito desta dissertação é o *Design Science Research* (DSR), é composto por seis fases distintas, a saber:

1. Identificação do problema – descrito nos primeiros temas do primeiro capítulo.
2. Definir objetivos da solução – neste passo é definido um conjunto de especificações que vai guiar a investigação e o qual o protótipo deve cumprir;
3. Desenho e desenvolvimento – nesta fase deve ser planeado todo o processo de desenvolvimento e implementar aplicando o plano definido;
4. Demonstração
5. Avaliação – procede-se a um conjunto vasto de testes e medições ao protótipo fabricado;
6. Comunicação - concluídas as fases anteriores e caso a avaliação seja positiva de modo a não requerer melhoramento do artefacto, procede-se com a escrita da dissertação

Uma vez que DSR é um processo cíclico, no fim do passo cinco, caso haja necessidade de melhorar o que foi desenvolvido, é preciso recuar até ao passo dois e repetir o processo de modo a ir ao encontro com as especificações definidas. O número de iterações não é fixo e varia consoante o que é concluído no processo de avaliação. Apenas quando é concluído que o protótipo apresenta todas as características definidas, é que se avança para a comunicação.

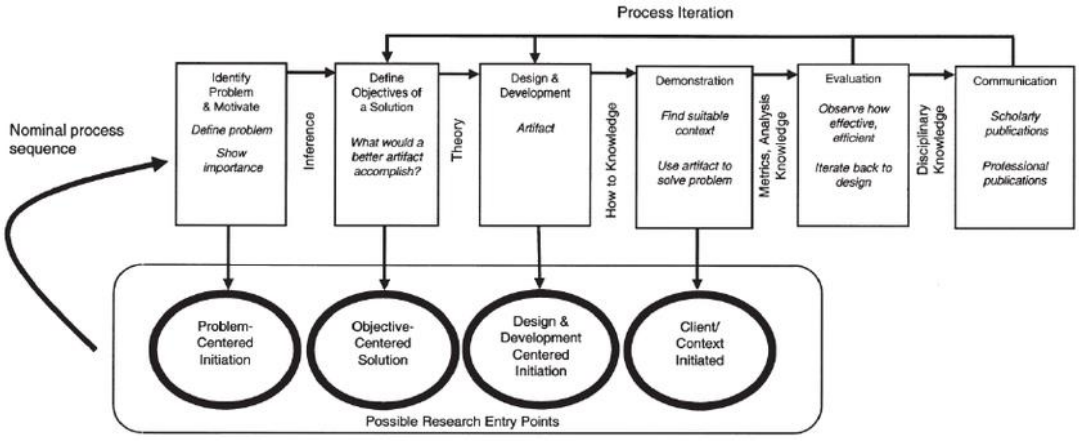

Figura 1 – *Design Research Adopted* (Peffers et al., 2007)

2 Revisão de literatura

Neste capítulo segue-se a análise ao estado de arte, onde irão ser referidos aspetos da plataforma Android e da segurança das aplicações para esse mesmo ambiente, será também referida uma explicação sucinta sobre vulnerabilidades e ferramentas de deteção das mesmas, assim como de métodos de cálculo de risco com enfase para o CVSS. Por fim, serão descritos alguns estudos que contribuíram, de certa forma, para o desenvolvimento deste projeto.

2.1 Plataforma Android

Android é um sistema operativo baseado no núcleo Linux, desenvolvido por um consorcio de programadores conhecido como *Open Handset Alliance*, sendo o principal colaborador da Google. O código do sistema Android é disponibilizado pela Google sob licença de código aberto, apesar de a maior parte dos dispositivos serem lançados com uma combinação de *software* livre e *software* privado. Inicialmente desenvolvido pela empresa Android, Inc. e posteriormente adquirida pela Google em 2005, a revelação oficial ao público foi em 2007.

Desde a publicação oficial do Android, houve muito interesse por parte de empresas, programadores e da audiência em geral sobre este novo tipo de *software*. Desde esse tempo até agora, o Android tem sido constantemente melhorado quer em termos de funcionalidades ou suporte de *hardware*, assim como em termos de extensibilidade como a adaptação a diferentes dispositivos para além de *smartphones*. Atualmente o Android já ganhou força em outros dispositivos de *hardware* assim como Televisões, *werables* e automóveis, por exemplo. O aumento do interesse da indústria deve-se a dois fatores essenciais: o facto do Android ser *open-source* e o seu modelo arquitetónico. Ao permitir o acesso ao seu código-fonte, torna-se possível analisá-lo completamente e entendê-lo. Deste modo os programadores têm uma melhor compreensão do sistema que permite a adaptação fácil para outros tipos de hardware, assim como corrigir erros, fazer melhoramentos consoante novas funcionalidades implementadas, entre outros. Por outro lado, a sua arquitetura baseada no *Kernel* de Linux também adiciona o uso do Linux na indústria de dispositivos móveis, permitindo assim dar uma vantagem a nível de conhecimentos sobre funcionalidades oferecidas pelo Linux. (Gandhewar & Sheikh, 2010)

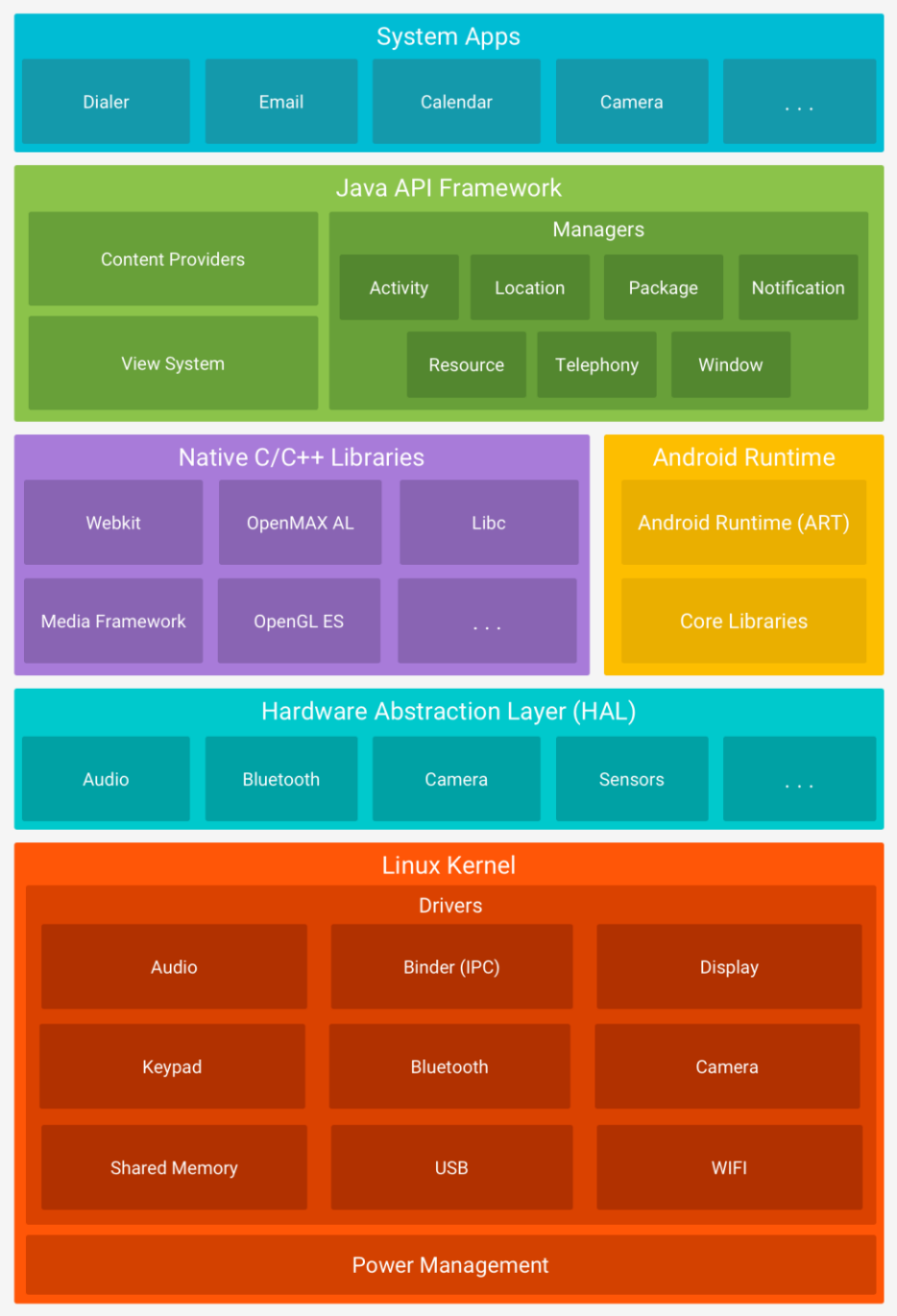

Figura 2 – Diagrama da Arquitetura Android (Android Developers, 2020)

A plataforma Android é constituída por um conjunto de componentes divididos em camadas. A arquitetura da plataforma pode ser visualizada na Figura *2*, assim como os respetivos componentes que constituem o *software.*

- **Aplicações do Sistema:** Nível mais alto do sistema e é o que fornece um conjunto de aplicações nativas que já vêm pré instaladas no dispositivo (sem serem instaladas pelo utilizador), estas aplicações não têm permissões especiais e podem ser desinstaladas se o utilizador quiser à exceção da aplicação de definições do sistema. Todas as aplicações são escritas com a linguagem Java;

- **Java API Framework:** é uma ferramenta de *software* utilizada para implementar estruturas standard nas aplicações. Também escritas na linguagem de programação Java, esta ferramenta é responsável por criar blocos de código durante o desenvolvimento de aplicações de modo a facilitar esse processo. O objetivo é que possa ser reutilizado código uma vez que existem funcionalidades que todas ou quase todas as aplicações as usem. Outro aspeto importante é que estas funcionalidades comunicam diretamente com o sistema operativo. Exemplo de funcionalidade: sistema de gestão de notificações que permitem mostrar alertas personalizados ou componentes de interfaces gráficas como listas, grelhas, caixas de texto, entre outros;
- **Android Runtime:** para dispositivos Android com versão 5.0 ou superior, cada aplicação é executada com uma instância do *Android Runtime* (ART). O ART foi desenhado para executar várias máquinas virtuais em dispositivos com capacidades de memória mais baixas, este executa ficheiros DEX, um formato bytecode implementado especialmente para Android com a característica de que oferece um consumo mínimo de memória;
- **Bibliotecas C/C++:** recursos do sistema Android que são desenvolvidas em C e C++, servem de apoio ao desenvolvimento de aplicações quando existe a necessidade de comunicar com camadas mais inferiores da arquitetura. o *Android Runtime* e o *Hardware Abstarction Layer* (HAL), são implementados por código nativo com recurso a bibliotecas nativas escritas em C e C++;
- **Camada de abstração de hardware: a** camada de abstração de hardware (HAL) fornece interfaces padrão que expõem as capacidades de hardware do dispositivo para a estrutura da API do Java de maior nível. A HAL consiste em módulos de bibliotecas, que implementam uma interface para um tipo específico de componente de hardware, como o módulo de câmara ou Bluetooth. Quando uma Framework API faz uma chamada para aceder ao hardware do dispositivo, o sistema Android carrega o módulo da biblioteca para este componente de hardware;
- **Núcleo do Linux:** na base da pilha arquitetónica da plataforma Android, encontra-se o núcleo do Linux (Linux *Kernel*). Desta forma, muitas funcionalidades do sistema operativo Android englobam funcionalidades do Linux assim como muitas características do sistema operativo de código aberto mais conhecido do mundo. Como por exemplo, cada aplicação num dispositivo Android, é um utilizador com certas permissões no sistema tal como um verdadeiro utilizador de uma máquina Linux. O ART confia no núcleo do Linux para cobrir funcionalidades como encadeamento e gerenciamento de memória de baixo nível. Usar um núcleo do Linux

permite que o Android aproveite os recursos de segurança principais e que os fabricantes dos dispositivos desenvolvam drivers de hardware para um *Kernel* conhecido.

No nível do sistema operativo, a plataforma Android fornece a segurança do *Kernel* Linux, bem como um recurso de comunicação segura entre processos (IPC) para permitir a comunicação segura entre aplicações em execução em diferentes processos. Esses recursos de segurança no nível do SO garantem que até mesmo o código nativo seja restringido pelo *Application Sandbox*. Quer esse código seja o resultado do comportamento da aplicação incluída ou uma exploração de uma vulnerabilidade da aplicação, o sistema é projetado para impedir que a aplicação nociva afete outras aplicações, o sistema Android ou o próprio dispositivo.

O núcleo do Linux fornece ao Android vários recursos de segurança importantes, como por exemplo:

- Um modelo de permissões com base no utilizador
- Isolamento de processos
- Mecanismo extensível para IPC seguro
- A capacidade de remover partes desnecessárias e potencialmente inseguras do *Kernel*

Tendo em conta que se trata de um sistema operativo multiutilizador, um objetivo fundamental de segurança do *Kernel* Linux é isolar os recursos dos utilizadores uns dos outros. Alguns exemplos retirados da documentação da *Android Developers* explicam bem a limitação de recursos a outros utilizadores do sistema.

- Evita que o utilizador A leia os arquivos do utilizador B
- Garante que o utilizador A não esgota a memória do utilizador B
- Garante que o utilizador A não esgota os recursos da CPU do utilizador B
- Garante que o utilizador A não esgota os dispositivos do utilizador B (por exemplo, telefonia, GPS e Bluetooth)

2.2 Componentes de uma aplicação Android

Os componentes de uma aplicação Android são os blocos de construção essenciais para o seu desenvolvimento. Cada componente é um ponto de entrada para o sistema ou um utilizador aceder à aplicação. Alguns componentes dependem uns dos outros. Existem quatro tipo de componentes: Atividades, Serviços, *Broadcast Receivers* e *Content Providers*. Cada componente tem um propósito diferente assim como um ciclo de vida diferente. Nesta secção irá ser explicado cada um dos componentes.

2.2.1 Atividades

Uma atividade é o ponto de entrada de interação com o utilizador. A atividade representa uma vista única com uma interface gráfica. Por exemplo, uma aplicação de *e-mails* pode ter uma atividade que mostra uma lista de novos *e-mails*, outra atividade que compõe um *e-mail* e outra ainda que lê *e-mails*. Embora essas atividades funcionem juntas para formar uma experiência coesa ao utilizador na aplicação de *e-mails*, elas são independentes entre si. Portanto, uma aplicação diferente pode iniciar qualquer uma dessas atividades (se a app de *e-mails* permitir). Por exemplo, uma aplicação que faça uso da câmara pode iniciar a atividade na app de *e-mails* que compõe o novo *e-mail* para que o utilizador compartilhe uma foto.

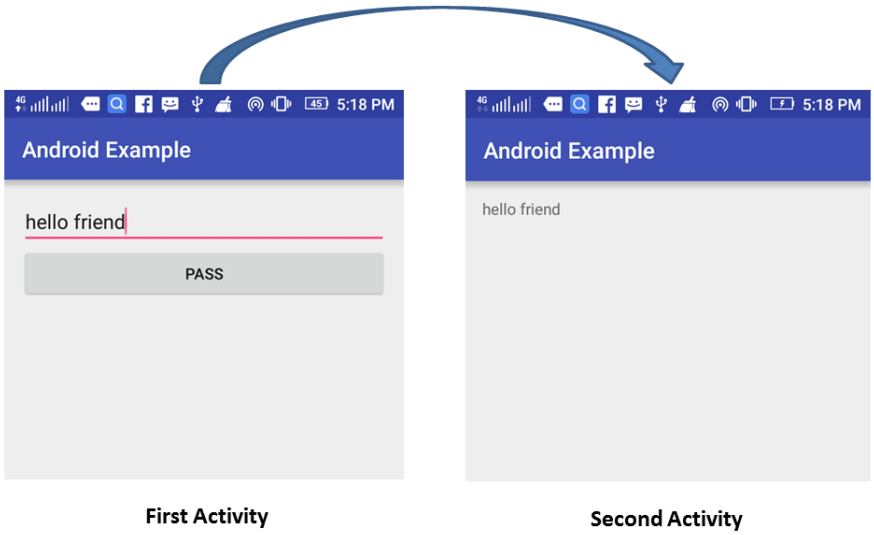

Figura 3 – Exemplo de duas atividades de uma aplicação Android (Android Developers, 2020)

2.2.2 Serviços

Os serviços são responsáveis por manter as aplicações em execução mesmo quando estas estão em segundo plano, seja qual for o motivo. Os serviços não apresentam interface tal como as atividades, no entanto podem chegar ao utilizador de forma direta através do seu dispositivo. Por exemplo, um serviço pode tocar música em segundo plano enquanto o utilizador está noutra aplicação diferente ou enquanto tem o seu ecrã bloqueado. Outro exemplo, é quando as aplicações fazem atualizações automáticas do software, normalmente isso acontece em segundo plano sem que o utilizador seja interrompido quando está a usar o seu dispositivo. Porém, no caso do serviço responsável por tocar música, a aplicação necessita informar ao sistema que precisa de ficar em primeiro plano, não através de uma atividade, mas sim através de uma notificação, desta forma o sistema sabe que o processo tem de permanecer ativo até o utilizador não desejar, quando remover a notificação por exemplo. Os serviços comuns em segundo plano, normalmente não são conhecidos pelo utilizador, desta forma o

sistema tem maior liberdade para gerir os processos de acordo com a RAM (*Random Access Memory*) disponível, por exemplo executar processos que tenham prioridade sobre outros.

Outro tipo de serviços em aplicações android são os serviços vinculados, estes são executados quando outras aplicações informam que precisam de usá-los. É como se fosse um serviço a fornecer uma API para outro processo. O sistema entende que existe uma dependência entre processos, ou seja, se o processo A for vinculado ao processo B, o sistema sabe que precisa manter B em execução de modo a que A consiga terminar a sua função. Os serviços são um elemento básico muito útil para o funcionamento das aplicações. Planos de fundo interativos, detetores de notificação, protetores de ecrã, métodos de entrada, serviços de acessibilidade e muitos outros recursos fundamentais do sistema são criados como serviços que as aplicações implementam e a que o sistema se vincula quando há a necessidade de execução.

2.2.3 Broadcast Receivers

Os *broadcast receivers* são componentes que fazem o sistema entregar eventos às aplicações sem que estas estejam em execução. Isto permite às aplicações responderem a anúncios de transmissão por todo o sistema. Por exemplo, aplicações podem programar alertas para notificar o utilizador sobre um evento futuro. Ao entregar essa notificação a um recetor de transmissão, a aplicação não precisa de permanecer em execução após a notificação chegar ao ecrã do dispositivo. Muitas transmissões têm origem no próprio sistema Android. Por exemplo, uma transmissão que informa para o ecrã se desligar, notificar quando a bateria está a ficar sem carga ou a captura de ecrã. Os *broadcast receivers* não apresentam nenhum tipo de interface gráfica tal como as atividades, mas eles criam notificações na barra de *status* com o objetivo de alertar o utilizador.

2.2.4 Content Providers

Fornecedores de conteúdo geram um conjunto compartilhado de dados da aplicação que podem ser armazenados no sistema de ficheiros, na base de dados *SQLite*, na *web* ou em qualquer local de armazenamento acessível à aplicação. Através do fornecedor de conteúdo, outras aplicações podem consultar ou até mesmo modificar dados, caso o fornecedor de conteúdo o permita. Por exemplo, o sistema Android oferece um fornecedor de conteúdo que gere os dados de contato do utilizador. Assim qualquer aplicação com as permissões adequadas pode consultar parte do fornecedor de conteúdo (como *ContactsContract.Data*) para ler e gravar informações sobre uma pessoa específica. É errado pensar nos fornecedores de conteúdo como uma abstração de uma base de dados porque existe suporte e APIs para esse caso. No entanto, de uma perspetiva de desenvolvimento de sistemas, eles

têm um propósito principal diferente. Para o sistema, o fornecedor de conteúdo é um ponto de entrada numa aplicação para a publicação de dados identificados por um URI.

- A atribuição de um URI não exige que a aplicação permaneça em execução. Dessa forma, os URIs podem persistir mesmo quando as suas aplicações estejam encerradas. Quando é necessário recuperar os dados da aplicação no URI correspondente, o sistema só precisa de garantir que uma aplicação correspondente esteja em execução.
- Os URIs também fornecem um importante modelo de segurança de controle preciso. Por exemplo, uma aplicação pode substituir o URI por uma imagem que esteja na área de transferência, mas deixar o *content provider* bloqueado para que outras aplicações não possam aceder livremente. Quando uma segunda aplicação tenta acessar o URI na área de transferência, o sistema pode permitir o acesso por meio de uma *concessão temporária da permissão do URI*. Assim é permitido o acesso aos dados somente por trás desse URI.

Um aspeto único do sistema Android é o facto de qualquer app poder iniciar um componente pertencente a outra app diferente. Por exemplo, quando se pretende desenvolver a função de captura de imagem fazendo uso da camara, como já existe outra aplicação que faz exatamente essa mesma função, o programador pode fazer uso dessa app para a sua que está a desenvolver em vez de implementar essa função de raiz. O Android permite iniciar a atividade da aplicação da camara ao ser chamada. Assim que a atividade é iniciada, todo o processo é gerido pela aplicação da camara.

Uma vez que no sistema android cada aplicação é executada em processos separados com permissões que restringem o acesso a outras aplicações, uma não pode aceder diretamente ao componente de outra. No entanto, o sistema Android permite que o componente de uma aplicação seja ativado por outra através de uma mensagem especifica, o *Intent.*

2.2.5 Intent

Três dos quatro tipos de componentes, atividades, serviços e *Broadcast receivers*, são ativados por mensagens assíncronas chamadas de *Intent's. Intent's* funcionam como mensageiros que pedem ações de outros componentes, quer sejam da mesma aplicação ou de outra diferente. Para os serviços e atividades, um *Intent* define a ação que vai tomar e pode especificar o URI dos dados onde vai atuar, entre outras coisas que o componente a ser iniciado tem de ter conhecimento. Por exemplo, um *Intent*, pode transmitir um pedido para abrir uma imagem ou uma página web. Também, é possível iniciar atividades para receber um resultado, os resultados provenientes dessa atividade também vêm sob a forma de *Intent.* Por exemplo, um *Intent* pode estar preparado para quando um utilizador quiser ter informações de um contacto, neste caso o *intente* retorna a URI que aponta para o contacto em questão.

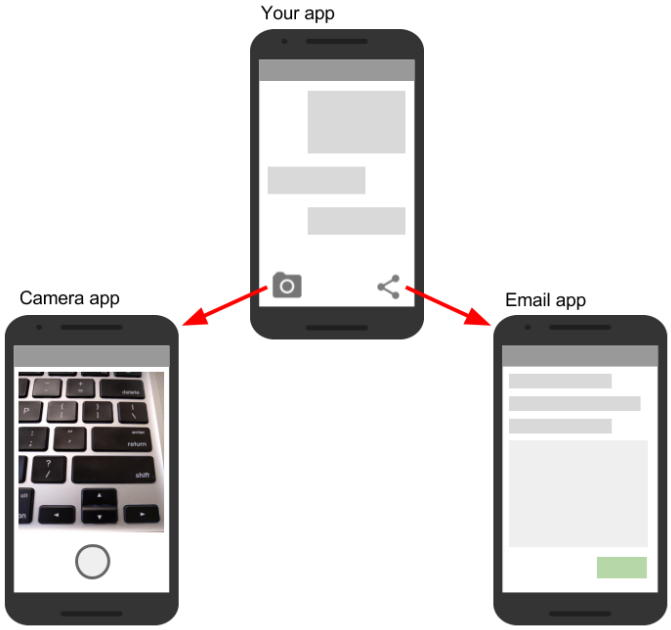

Figura 4 – Exemplo ilustrativo de uma aplicação que pode iniciar atividades de outras aplicações diferentes através de *Intent's* (2.1: Activities and intents, 2020)

2.2.6 Android Package

Android Package (APK) é um arquivo de pacote destinado ao sistema operativo Android. É uma sigla em inglês da palavra *Android Application Pack*. Ele pode ser comparado com os arquivos proprietários de instalação de software do Windows, como o ".exe" ou o ".msi". Acontece que o APK tem um formato semelhante a compressão que é feita pelo ZIP, e no seu interior ficam todos os arquivos necessários para a instalação de apps e jogos. Para criar um arquivo APK, todo o código-fonte da aplicação é compilado e depois empacotado em um único arquivo, usando ferramentas como o programa aapt, dx.bat (encontradas no Android SDK), javac, usado na compilação das classes java presente no código-fonte, e outros programas específicos. O APK é um formato de compressão ZIP baseado no formato JAR.(Wikipedia, 2020).

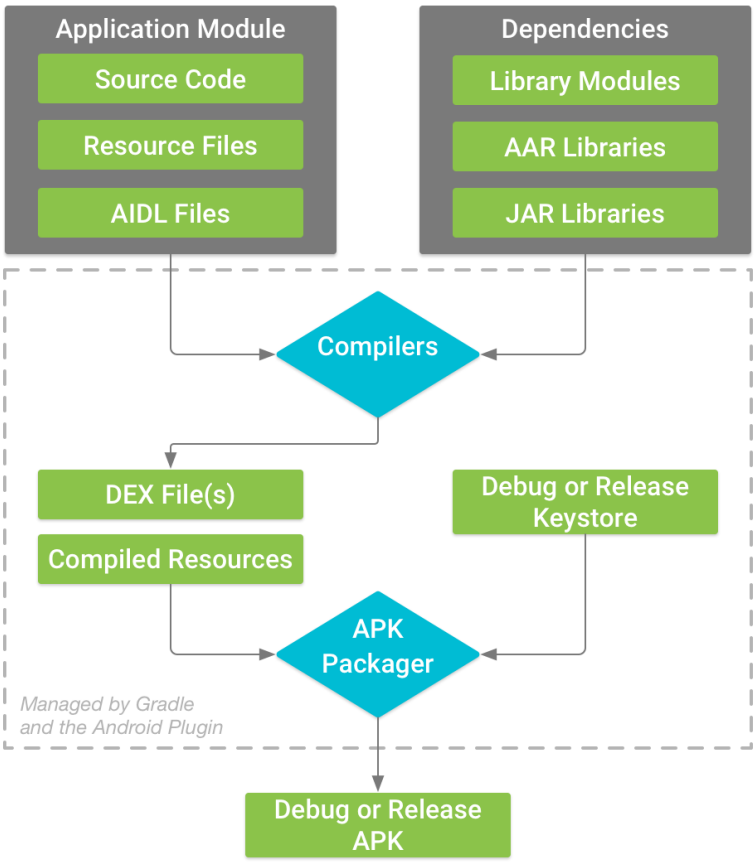

Figura 5 - processo de construção de um APK (Android Developers, 2020)

O processo normal da construção de um APK, tal e qual como a Figura 5 mostra, segue vários passos. Primeiro, os compiladores convertem o código fonte para ficheiros DEX (*Dalvik Executable*), estes ficheiros incluem o *bytecode* que é executado em dispositivos Android, e também convertem os recursos necessários para ficheiros compilados. Posteriormente, o *APK Packager* combina os ficheiros DEX e os recursos compilados num único APK. O APK não pode ser instalado num dispositivo Android sem ser assinado, o *APK Packager* também é responsável por essa assinatura podendo assinar com um certificado *debug* ou através da *keystore* de *release*. Normalmente a assinatura com *debug keystore* é para quando apenas se pretende testar as aplicações, enquanto que a *release keystore* é um certificado mais seguro que um sistema de chaves privadas e públicas encriptadas, apenas o detentor da aplicação desenvolvida tem acesso à chave privada, deste modo, apenas o detentor dessa chave pode publicar a sua aplicação nas lojas. (Android Developers, 2020)

2.3 Segurança nas aplicações Android

O Android tem estado disponível desde 2007, o primeiro dispositivo *Android-ready* foi publicado em outubro de 2008. Desde então, o crescimento da plataforma Android tem sido fenomenal (Fang et al., 2014). O sistema Android restringe a interação de aplicações para suas API's correndo as aplicações

com a sua própria identificação de utilizador. Contudo esta interação controlada tem características que são mais vantajosas para a segurança, mas desenvolver uma aplicação segura não é assim tão linear (Robinson & Weir, 2015).

2.3.1 Modelo de permissões

O sistema Android usa um simples modelo de permissões para restringir acesso de aplicações a recursos sensíveis como SMS ou informação externa armazenada num cartão de memória, por exemplo. O acesso a recursos sensíveis pode originar perdas financeiras. Por exemplo, um *malware* para o sistema Android pode passar o limite de mensagens de um utilizador ou pode consumir dados de internet sem que um utilizador se aperceba. Também podem ocorrer fugas de informação privada de utilizadores quando é permitido o acesso a certos recursos como contatos, emails e cartões de crédito. Programadores de aplicações podem aproveitar vários sensores de smartphone como GPS, câmaras, microfones e desenvolver aplicações que fazem mais do que o seu principal propósito como copiar informação privada de forma discreta (Fang et al., 2014).

A privacidade e a segurança, nas versões mais antigas do sistema Android, eram geridas no período de instalação de uma aplicação quando esta pedia permissões ao sistema. Cada aplicação declarava quais as permissões que pretendia usar e o utilizador era notificado durante a instalação. Se o utilizador não aceitasse as permissões de uma certa aplicação, o processo de instalação podia ser interrompido. Nas versões mais recentes, este processo ocorre depois da instalação, quando a aplicação já está a correr. Deste modo a aplicação pode estar sempre a pedir permissões cada vez que é aberta.

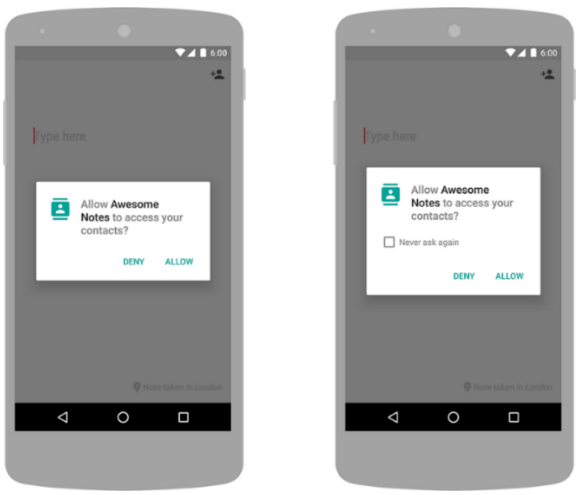

Figura 6 – exemplo de pedido de permissão para aceder aos contactos

Permissões no período de instalação podem fornecer aos seus utilizadores controlo sobre a sua privacidade e reduzir o impacto de *bugs* e vulnerabilidades nas aplicações. Contudo, um sistema de permissões durante a instalação é ineficaz se os programadores constantemente requisitarem mais permissões do que o normal. Aplicações que solicitam mais permissões do que as necessárias, expõem os utilizadores a avisos desnecessários de permissões assim como aumentam o impacto de um *bug* ou de uma vulnerabilidade (Felt et al., 2011). Utilizadores de Android, normalmente, adquirem aplicações na Google Play Store, Amazon Appstore ou na Aptoide. A qualidade e confiança das aplicações é muito variável de aplicação para aplicação, por isso a Android denota todas as aplicações como potencialmente vulneráveis ou maliciosas. Cada aplicação corre num processo com um identificador de utilizador poucos privilégios e as aplicações conseguem apenas aceder aos próprios ficheiros por defeito. As permissões no sistema Android são de quatro tipos: *normal, signature, signatureOrSystem e dangerous.* Atualmente existem 30 permissões com um nível de proteção *"dangerous"* (Android Developers, 2020).

O acesso excessivo de permissões por si só não é um problema, o problema é quando uma aplicação contém código malicioso (*malware*), e aproveita o acesso facilitado para aceder a conteúdo sensível. O primeiro *malware* identificado para Android foi em 2010, o *FakePlayer,* e é um exemplo de uma aplicação que usa permissões para enviar SMS de forma maliciosa sem que o utilizador se aperceba (Zhou & Jiang, 2012).

2.3.2 Comunicação segura

Ao longo do tempo, telemóveis têm evoluindo de uma forma muito significativa e o sistema de comunicação dos mesmos já não fica apenas pelo envio de SMS ou de chamadas. Agora os telemóveis são mais sofisticados e podem correr software desenvolvido por qualquer entidade terceira. Os utilizadores também podem instalar qualquer aplicação mesmo que não tenha sido adquirida numa loja oficial de aplicações.

O modelo de comunicação em aplicações Android promove o desenvolvimento com dependências externas de forma a poder enriquecer as aplicações. Com isto, programadores Android conseguem aproveitar dados e serviços fornecidos por outras aplicações, tal e qual como os *Intents,* e mesmo assim a aplicação desenvolvida pode passar a impressão de ser simples e sem qualquer ligação com terceiros (Chin et al., 2011). Vejamos um exemplo, uma aplicação de avaliações de restaurantes pode pedir a outra aplicação para disponibilizar o *website* do restaurante, para fornecer um mapa com a localização do restaurante ou para telefonar para o mesmo. Este tipo de modelo de comunicação reduz a carga de trabalho de um programador e promove a reutilização de funcionalidades. Isto é possível porque o sistema Android divide as aplicações por componentes e fornece um canal de

mensagens para que esses componentes comuniquem entre si quer dentro ou fora das fronteiras de uma aplicação (Chin et al., 2011).

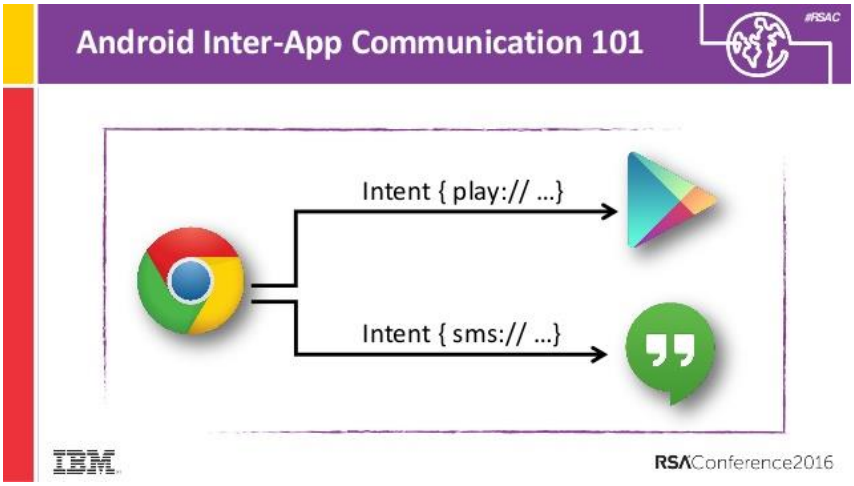

Figura 7 – Exemplo ilustrativo de comunicação entre aplicações

Este sistema de comunicação pode-se tornar num terreno de ataque se for usado de forma incorreta. Se o componente remetente do envio de uma mensagem não identificar devidamente o componente destinatário, então um atacante pode intercetar uma mensagem e assim comprometer a confidencialidade e integridade. Se um componente não restringir para quem deve enviar mensagens, então um atacante pode injetar mensagens maliciosas para outros componentes(Chin et al., 2011).

2.3.3 Armazenamento de dados no sistema Android

De acordo com o OWASP (*Open Web Application Security Project*), armazenamento inseguro de dados é um dos dez maiores problemas de segurança em *smartphones* quando a informação sensível pode ser revelada caso não seja protegida de forma correta. Programadores com menos experiência assumem que o acesso à memória interna do dispositivo não pode ser acedido, mas um atacante com acesso físico ao dispositivo, pode conectá-lo ao computador e retirar informação sensível e pessoal de um utilizador. Além disso, aplicações maliciosas e aplicações vulneráveis podem ser uma ameaça para a fuga de informação. Um "cavalo de Troia" escondido numa aplicação legitima pode exercer atividades de interceção com o objetivo de roubar informações ao utilizador. Ainda mais, atacantes que querem acesso aos dados armazenados no dispositivo podem explorar aplicações vulneráveis, como aplicações que fazem um uso excessivo de permissões (Altuwaijri & Ghouzali, 2020).

No sistema Android, existem três alternativas de armazenamento de dados: "*system*", "*application-specific*" e "*public*". O armazenamento interno do sistema ("*system*") é o catálogo onde o

sistema operativo Android é localizado e assegurado pelo componente de acesso do Linux. O armazenamento "*application specific"* é um espaço reservado que está sob controlo de uma aplicação e apenas pode ser lido ou escrito por essa mesma aplicação, geralmente montado em *"/data/".* Cada aplicação tem uma diretoria privada que normalmente armazena informação sensível como as credenciais de *login*. A outra alternativa de armazenamento em Android, é partilhada publicamente, normalmente montada em *"/sdcard/"* ou *"/mnt/sdcard/".* Este tipo de armazenamento é utilizado para partilhar informações entre aplicações. É geralmente usado para guardar ficheiros de multimédia, criados pelo uso da câmara ou do microfone, porém permite descarregar ficheiros, não só multimédia, quando o dispositivo está conectado ao computador via USB. Mais ainda, o Android permite o uso de cartões SD removíveis, em termos de funcionalidade é igual ao cartão SD interno (Altuwaijri & Ghouzali, 2020).

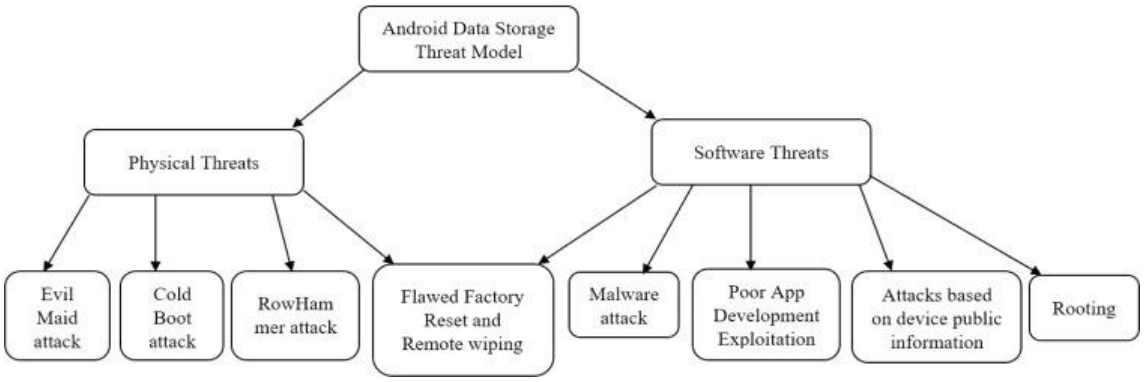

Figura 8 – Modelo de ameaça para o armazenamento de dados em Android (Altuwaijri & Ghouzali, 2020)

De acordo com a Figura *8* podemos ver os vários tipos de ataques que podem ser realizados para explorar o armazenamento de dados no sistema Android. Os ataques podem ser realizados com acesso físico ao dispositivo, em que os dados dos utilizadores permanecem no dispositivo muito tempo depois dos utilizadores os deixarem de usar, caso não tenham apagado os dados. A exploração física de componentes como a memória RAM (Random Access Memory) ou o disco de armazenamento, pode levar à descoberta de informação muito sensível como passwords. Outros tipos de ataques podem ser feitos sem o acesso físico ao dispositivo, caracterizados como ataques de *software* onde uma das subcategorias é o desenvolvimento inseguro de aplicações, que é o foco principal desta dissertação (Altuwaijri & Ghouzali, 2020).

2.4 Vulnerabilidades

A avaliação de vulnerabilidades tem um papel importante para a segurança e gestão de risco. Uma vulnerabilidade representa uma fraqueza ou uma falha, originalmente causada no desenho ou na

implementação que pode ser explorada para violar a política de segurança de um sistema. Num sistema de informação, falhas e fraquezas podem originar acessos não autorizados por terceiros com o objetivo de danificar ou comprometer o funcionamento do mesmo. De modo a avaliar uma vulnerabilidade, é necessário definir métricas de segurança para medir o nível de severidade de uma vulnerabilidade baseado numa abordagem científica, sistemática e quantitativa. Sem métricas de segurança bem definidas, empresas e organizações encontram dificuldades em comprar e selecionar diferentes opções de segurança que de facto resolvam o problema em causa (J. A. Wang et al., 2009).

2.4.1 OWASP Mobile Security Project

O *Open Web Application Security Project (OWASP),* é uma organização sem fins lucrativos que se dedica ao melhoramento da segurança em *software.* Através de projetos comunitários abertos ao público incluindo o código fonte das mesmas, conteúdos educacionais como por exemplo conferências de formação pedagógica, esta organização serve as necessidades de programadores e de profissionais na área das TI para segurar a web. Da mesma forma, o mesmo conceito também foi adaptado com o crescimento da utilização e criação de aplicações em dispositivos móveis. O *OWASP Mobile Security Project* também fornece aos programadores e especialistas de segurança ferramentas e conteúdos de apoio à segurança para aplicações móveis. O seu objetivo é classificar riscos de segurança em ambiente móvel e fornecer controlos de desenvolvimento para que o impacto ou a probabilidade de exploração de falhas de segurança seja reduzida (OWASP, 2019).

Atualmente o *OWASP Mobile Security Project* fornece os seguintes recursos de apoio:

- *Android CK Project* – uma ferramenta desenvolvida em *Python* para análise forense para ambiente Android
- *Damn Vulnerable iOS Application* – Uma aplicação para iOS que é altamente vulnerável. O seu objetivo é puramente didático, entusiastas e estudantes podem usar esta aplicação para testar as suas habilidades de testes de intrusão.
- *iGoat Tool Project* – uma ferramenta de aprendizagem para programadores de iOS.
- *Mobile Application Security Verification Standard* – Um modelo standard que enumera os requisitos necessários de segurança de uma aplicação móvel para que seja considerada segura. Pode ser utilizada por arquitetos ou programadores de software que procuram desenvolver aplicações de forma segura.
- *Mobile Security Testing Guide* – Um manual explicativo para testes de segurança e engenharia reversa para aplicações móveis, iOS e Android.
- *Mobile Security Checklist* – Uma lista que ajuda no mapeamento e no nivelamento dos requisitos baseada na *Mobile Application Security Verification*.

- *MSTG Hacking Playground* - Uma plataforma com várias aplicações vulneráveis para iOS e Android.
- *Seraphimdroid* – Uma aplicação de proteção de dados para Android

Uma das formas mais valorizadas para categorizar vulnerabilidades em ambientes móveis, concebida pela *OWASP Mobile Security Project*, é o *OWASP Mobile Top 10*. Consiste numa lista resumida dos riscos que afetam aplicações móveis. Atualmente existem duas versões, uma datada em 2014 e outra em 2016, que é a mais recente. A lista atual é composta pelas seguintes categorias de risco:

- M1: Uso impróprio da plataforma
- M2: Armazenamento inseguro de dados
- M3: Comunicação insegura
- M4: Autenticação insegura
- M5: Criptografia insuficiente
- M6: Autorização insegura
- M7: Má qualidade de código
- M8: Violação de código
- M9: Engenharia reversa
- M10: Funcionalidade não desejada

2.4.2 Common vulnerabilities and Exposures

A atual lista de CVE´s ou *Common Vulnerabilities and Exposures*, concebida por David E. Mann e por Steven M. Christey proveio de uma investigação realizada em 1999. A comunidade de cybersegurança endorsa a importância do CVE e atualmente muitas organizações incorporam os CVE´s nos seus produtos e serviços. O CVE é uma lista pública de entradas onde cada uma contém um número de identificação, uma descrição e pelo menos uma referência pública com o objetivo de formalizar vulnerabilidades de cybersegurança de modo a que sejam compreendidas por toda a parte do mundo (CVE, 2018).

2.4.3 National Vulnerability Database

A *National Vulnerability Database,* é uma base de dados pública e oficial que contém entradas de CVE. Criada pelo governo dos Estados Unidos da América, é considerada uma das mais confiáveis base de dados de vulnerabilidades pela comunidade de cybersegurança. Várias investigações já foram

realizadas com o *dataset* fornecido pela NVD (Petraityte et al., 2018). Em 2019 existiam mais dez mil CVEs na base de dados (NIST, 2019).

2.5 Metodologias de cálculo de risco

Nesta secção são apresentadas algumas técnicas de cálculo e avaliação de risco. Existem abordagens diferentes para o mesmo objetivo que já foram usadas em casos reais. Deste modo é reforçada a importância de analisar várias metodologias que podem ser a solução para o cálculo de risco e ameaça durante o desenvolvimento de aplicações Android.

2.5.1 Damage, Reproducibility, Exploitability, Affected users, Discoverability

DREAD (*Damage, Reproducibility, Exploitability, Affected users, Discoverability*), proposto pela Microsoft, é um acrónimo que representa cinco critérios de avaliação de ameaças e para cada um desses critérios é atribuído um nível de 1 a 10, onde níveis maiores representam ameaças mais sérias. O cálculo do DREAD é simplesmente a média da soma dos níveis atribuídos a cada critério.

- ***Damage*** - Avaliação do dano que um ataque pode causar. Dano pode incluir perda de dados, falha de hardware ou de dispositivos, redução de performance ou qualquer outro tipo de dano que possa afetar um dispositivo ou o seu sistema operativo.
- ***Reproducibility –*** Medição da reprodutibilidade de um ataque sucedido. Um ataque que seja facilmente reproduzido é mais provável que seja explorado. Por exemplo, ameaças a programas que estão instalados por defeito num dispositivo.
- ***Exploitability*** Avaliação do esforço, conhecimento e nível de acesso necessário para elaborar um ataque. Se uma vulnerabilidade for facilmente explorada por alguém com poucos conhecimentos e remotamente, maior o risco de ameaça.
- ***Affected* Users**. O número de utilizadores que podem ser afetados por um ataque é um fator importante na avaliação de uma ameaça. Quantos mais utilizadores possam ser afetados maior o nível deste critério.
- ***Discoverability*** Avaliação da probabilidade de uma vulnerabilidade poder ser explorada. Sendo uma medida difícil de prever, a Microsoft aconselha a atribuição de um valor alto assumindo que uma vulnerabilidade pode ser sempre explorada eventualmente.

Fórmula para o cálculo de DREAD (Microsoft, 2018):

$$DREAD = \frac{D + R + E + A + D}{5}$$

2.5.2 The OWASP Risk Rating Methodology

A metodologia de cálculo de risco pela OWASP é apresentada de modo a estimar riscos de vulnerabilidades associados a produtos de software. Esta organização apresenta um método diferente onde o risco se baseia na probabilidade de uma vulnerabilidade ser explorada multiplicando pelo impacto que ela possa causar, quer a nível técnico quer a nível de negócio (OWASP, 2019). Por exemplo, se uma vulnerabilidade for muita facilmente explorável e se, ao ser explorada, causar uma falha técnica grave que afete o funcionamento de uma empresa de forma negativa, o nível de risco é alto. De modo a avaliar o risco com este método, a OWASP definiu 6 passos que devem ser considerados:

1. Identificação do Risco
2. Fatores para estimar o quão explorável pode ser uma vulnerabilidade
3. Fatores para estimar o impacto
4. Determinar a severidade do risco
5. Priorização de mitigação
6. Adaptar o modelo de risco para o modelo de negócio

A fórmula de calculo é dada da seguinte forma (OWASP, 2019):

$$Risk = Likelihood * Impact$$

Onde *"Likelihood"* é o nível de quão explorável pode ser uma vulnerabilidade, esta variável apresenta 3 níveis: *Low* [0,3[; *Medium* [3,6[; *High* [6,9]. *Impact* corresponde ao impacto que pode ser causado pela exploração de uma vulnerabilidade. O cálculo do impacto é mais complexo, e é dividido por dois fatores, técnico e de negócio. Para os fatores técnicos, é preciso determinar o nível de perda de confidencialidade, o nível de perda de integridade, o nível de perda de disponibilidade e o nível de rastreamento. Para o impacto de negócio é preciso ter em conta consequências como perdas financeiras, perda de reputação, violação de privacidade e nível de exposição causado por não conformidades.

O autor (Rao et al., 2010) que aplicou este método de avaliação de risco, mostra que é uma contribuição positiva para a gestão de risco de um produto de software.

2.5.3 Common Vulnerability Scoring System

O *Common Vulnerability Scoring System* ou simplesmente CVSS, foi desenhado pela *National Institute of Standards and Technology* mas é atualmente mantido e desenvolvido pelo *Special Interest Group* que trabalha sobre orientação da *Forum for Incident Response and Security Teams* (FIRST). Atualmente o CVSS já conta com três versões, sendo o atual CVSSv3. O governo dos Estados Unidos da América usa

o CVSS na sua base de dados nacional de vulnerabilidades (*National Vulnerability Database*) e para o programa de validação de produtos no protocolo de automação de conteúdos de segurança. O CVSS também é adotado por dezenas de fornecedores de serviços de software (Scarfone & Mell, 2009).
Como já foi referido num dos capítulos acima, o CVSS fornece uma ferramenta para quantificar o nível de severidade e risco de uma vulnerabilidade no ambiente de computação. As métricas do CVSS dividem-se em três grupos:

- Métricas Base, medem as características fundamentais e intrínsecas de uma vulnerabilidade que não se altera consoante o tempo ou ambiente.
- Métricas Temporais, medem atributos de uma vulnerabilidade que muda consoante o tempo, mas não consoante o ambiente.
- Métricas Ambientais, medem as características de vulnerabilidades que são relevantes e únicas para um determinado ambiente.

Existem seis métricas base que capturam as características mais fundamentais de uma vulnerabilidade:

1. *Access Vector* (AV): mede como uma vulnerabilidade pode ser explorada, localmente ou remotamente. Quanto mais remoto estiver o atacante a explorar a vulnerabilidade, mais alto é o nivel desta métrica.
2. *Access Complexity* (AC): Mede a complexidade de um ataque necessário para explorar a vulnerabilidade. Quanto menos complexo for o ataque, mais alto é o nível desta métrica.
3. *Authentication* (Au): Mede o número de vezes que um atacante deve autenticar um alvo para explorar a vulnerabilidade. Quanto menos autenticações necessárias, mais alto no nível desta vulnerabilidade.
4. *Confidentiality Impact* (CC): Mede o impacto na confidencialidade de uma vulnerabilidade explorada com sucesso. O impacto de alta confidencialidade aumenta o nível de severidade da vulnerabilidade.
5. *Integrity Impact* (IC): Mede o impacto na integridade de uma vulnerabilidade explorada com sucesso. Quanto maior o impacto de integridade maior o score da vulnerabilidade.
6. *Availability Impact* (AC): Mede o impacto na disponibilidade de uma vulnerabilidade explorada com sucesso. Quanto maior o impacto de disponibilidade, maior o score da vulnerabilidade.

As métricas temporais no CVSS representam as características dependentes do tempo numa vulnerabilidade, incluindo exploração em termos dos seus detalhes técnicos, o estado de remediação de uma vulnerabilidade e a disponibilidade do código ou das técnicas de exploração. As métricas ambientais representam as características especificas de implementação e ambiente de uma dada vulnerabilidade.

O processo de cálculo de um score começa primeiro com o cálculo das métricas base de acordo com a equação, onde é equacionado um valor que pode variar de zero a dez e cria um vetor. O vetor é representado por uma cadeia de carateres que contém os valores atribuídos a cada métrica base, isto tem como objetivo comunicar exatamente como o cálculo é derivado para cada vulnerabilidade. Opcionalmente, o cálculo base pode ser redefinido ao assumir as métricas temporais e/ou ambientais. Se for aplicada a métrica temporal, a equação irá combinar as métricas temporais com o cálculo base para calcular um valor temporal de zero a dez. Do mesmo modo, se for aplicada a métrica ambiental a equação também irá combinar as métricas base com as ambientais e calcular um valor de zero a dez.

2.6 Ferramentas de análise de Vulnerabilidades

Nesta secção serão apresentadas várias ferramentas de análise de vulnerabilidades para aplicações Android. As ferramentas aqui apresentadas são trabalhos de outros autores e o código fonte é aberto ao público. A análise é feita através de engenharia reversa sobre o ficheiro APK submetido e apenas é analisado o código fonte resultado da descompilação do ficheiro, por outras palavras, as ferramentas aqui apresentadas apenas fazem análise estática.

2.6.1 Androbugs

O Androbugs é uma *framework* de análise de vulnerabilidades para o sistema Android criada por YuChen Lin. A ferramenta usa o AndroGuard para efeitos de engenharia reversa, mas a deteção de vulnerabilidades é feita pelo Androbugs. Uma das principais características desta ferramenta é o facto de no final de uma análise, o resultado gerado é dedicado aos programadores de forma a poder ajudá-los a resolver os problemas encontrados. O seu funcionamento foca-se em encontrar más práticas de código, comandos potencialmente perigosos e trechos de código que comprometem a segurança da aplicação e tornando-a vulnerável a ataques. Também verifica se uma aplicação já foi alterada por "*repackaging*".

O Androbugs é uma ferramenta que apresenta um mecanismo otimizado e rápido para a análise de vulnerabilidades. A leitura do código gerado pela descompilação do APK é focada em encontrar "*strings*" e palavras chave que contenham informação relevante que apontam para uma

vulnerabilidade. A ferramenta também aposta numa análise rápida porque não analisa o código linha a linha, focando-se apenas no nome das funções e caso encontre algo suspeito então analisa o código dessa mesma função. Deste modo, uma análise com esta ferramenta é relativamente rápida.

Por fim, ao concluir a análise é criado um relatório com as vulnerabilidades encontradas, estas apresentam-se classificadas por severidade e é apresentado uma proposta de resolução ou um conselho de boa prática para resolver o problema.

2.6.2 DroidStatX

A ferramenta DroidStatX é outra ferramenta de análise estática para deteção de vulnerabilidades. Criada em 2015 por Claúdio André, a ferramenta diferencia-se pelo facto de quando encontra vulnerabilidades, classifica-as de acordo com o seu nível de *OWASP Mobile TOP 10 2016*. Desta forma, é possível associar certas características às vulnerabilidades ao terem esse grupo associado. Desenvolvido usando a linguagem de programação *Python*, após a análise de um APK a ferramenta gera um mapa em formato *Xmind* que reúne toda a informação e todas as evidências de possíveis vulnerabilidades identificadas. O mapa gerado tem como objetivo assistir "*penetration testers*" ao cobrir todos os aspetos importantes durante uma auditoria de segurança de um APK.

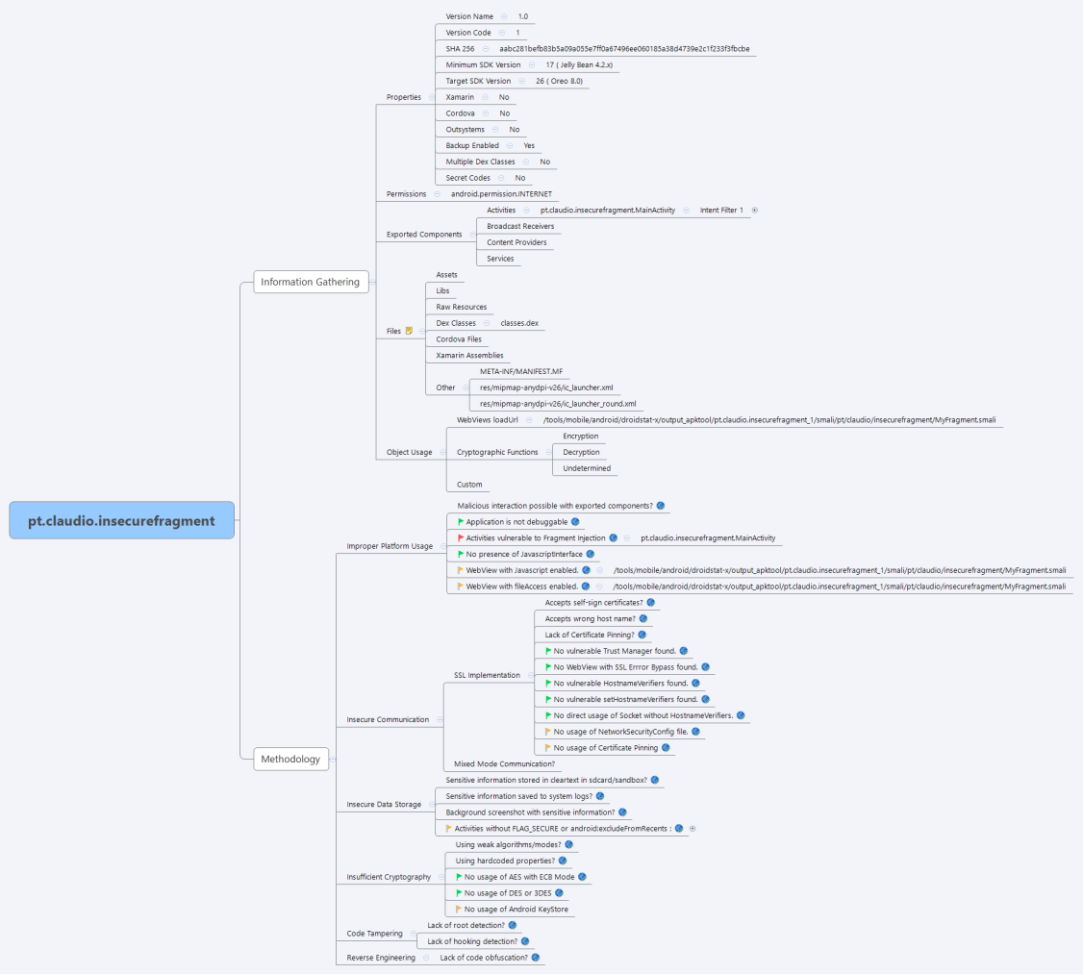

Figura 9 – Exemplo de resultado em Xmind gerado pelo DroidStatX

2.6.3 SUPER Android Analyzer

A ferramenta SUPER Android Analyzer (*Secure Unfied Powerful and Extensible Rust Android Analyzer*) tal como as referidas anteriormente, também analisa ficheiros APK em busca de vulnerabilidades. É uma ferramenta multiplataforma (Linux, MacOS X e Windows). De acordo com os criadores do SUPER, as outras ferramentas são incompletas e apresentam muitos problemas, ainda referem que o facto de serem desenvolvidas em *Java* ou *Python* e de não serem extensíveis, não sendo apropriadas para empresas e negócios.

A abordagem usada no desenvolvimento desta ferramenta difere de outras. Em primeiro lugar a equipa decidiu desenvolver usando a linguagem *Rust*, a linguagem desenvolvida pela *Mozilla Foundation*, que oferece várias funcionalidades e expressões regulares, manipulação de ficheiros entre outras. Também o *Rust* permite criar software seguro que não depende de descompiladores como JVM (*Java Virtual Machine*) e JIT (descompilador *Just-In-Time*). Erros como "*stack overflows*", "*segmentation faults*", entre outros erros, não são diretamente possíveis de acontecer. Desta forma, é possível fazer uma análise eficaz dando a opção de automatizá-la em grande volume.

Entretanto, a segunda exclusividade desta ferramenta é facto de ela ser cem por cento extensível. Todas as regras definidas para a deteção de vulnerabilidades, estão agregadas num único ficheiro (rules.json), assim cada organização ou cada utilizador da ferramenta pode criar os seus conjuntos de regras e adequar às suas necessidades. Também é uma ferramenta modular, programadores podem facilmente adicionar novas funcionalidades e por fim, também podem personalizar o modo de apresentação dos resultados.

Por fim, e à semelhança das outras ferramentas, o SUPER, devolve indicações detalhadas sobre as vulnerabilidades apresentado uma solução para mitigá-las caso exista. Também é indicada a localização da vulnerabilidade no código gerado, linha de código e ficheiro (Github, 2020).

2.6.4 QARK

O QARK *ou Quick Android Review Kit* é uma ferramenta de análise estática para a deteção de vulnerabilidades criada pela equipa de cybersegurança do LinkedIn, a *LinkedIn's House Security*. Tony Trummer e Tushar Dalvi em 2015 anunciaram esta nova ferramenta no Black Hat USA 2015 e DefCon23. A ferramenta foi desenhada para ser utilizada de forma gratuita pela comunidade, estando disponível para *download* no seu repositório GitHub. Tal como as outras ferramentas, o QARK também tem como objetivo educar programadores e ajudar profissionais de segurança a identificar vulnerabilidades e potenciais riscos relacionados com aplicações Android, oferecendo descrições concisas sobre os problemas e com ligações para referências fiáveis.

O QARK também tenta oferecer comandos ADB (*Android Debug Bridge*) dinamicamente gerados para adicionar validação a potenciais vulnerabilidades que foram efetivamente detetadas. Ainda vai criar dinamicamente uma aplicação personalizada de testes na forma de APK pronto a ser utilizado, desenhado especificamente para demonstrar os potenciais problemas que o QARK deteta.

No entanto, o QARK foi originalmente desenhado como uma biblioteca de apoio a teste manual, mas cresceu até se tornar numa ferramenta completa de testes. De acordo com os criadores, a ferramenta é bastante útil para organizações e empresas, mas recomenda que as mesmas façam revisões de segurança manual para as suas aplicações por três razões significativas: primeiro, análise estática ao código; segundo, atualização e manutenção; terceiro, nenhuma ferramenta é perfeita.
Concluindo, o QARK é uma ferramenta popular na comunidade GitHub e pelos profissionais de segurança Android e continua a receber suporte da equipa de desenvolvimento. A equipa foca-se bastante em minimizar o número de falsos positivos/negativos, em completar a habilidade de automaticamente verificar vulnerabilidades testando o APK que foi criado, corrigir *bugs* e adicionar funcionalidade para o Windows.

2.7 Trabalhos Relacionados

No seguimento da investigação desta solução foi tido em conta alguns trabalhos relacionados com o cálculo de risco para aplicações móveis. Embora as metodologias sejam diferentes daquilo que se pretende, trabalhos idênticos são relevantes para esta dissertação.

2.7.1 Previsão automática de score CVSS recorrendo à análise da descrição das vulnerabilidades

O trabalho investigado por (Khazaei et al., 2016), apresenta uma forma automática de atribuição de um nível CVSS a vulnerabilidades apenas analisando as suas descrições. Através de algoritmos de *Text Mining* com o objetivo de encontrar padrões e semelhanças entre vulnerabilidades para depois aplicar algoritmos de classificação.

Para conseguirem aplicar algoritmos de previsão e de *text mining*, os autores recorreram a duas bases de dados públicas que contêm vulnerabilidades e detalhes das mesmas, o CVE e o OSDVB, agregando as duas bases de dados e analisando as descrições das vulnerabilidades, obtiveram 19320 entradas no total. Tendo em conta que as descrições são textos com uma linguagem compreendida por humanos, contêm muitas palavras comuns com conjunções, advérbios, vogais e proposições. Essas palavras são conhecidas como *stopping-words.* Depois de removerem essas palavras reduziram o número para 8965 entradas.

Tendo agrupado o número de amostras para a experiência, os autores aplicaram dois algoritmos de classificação, *Support Vector Machine* (SVM) e *Random-Forest*, sendo os métodos mais confiados e usados para previsão (Khazaei et al., 2016), também foram usados sistemas de lógica difusa.

O melhor resultado foi obtido com o sistema de lógica difusa, sendo capaz de prever um score com oitenta e oito por cento de precisão.

2.7.2 DroidRisk

Os autores (Y. Wang et al., 2013) apresentam o *DroidRisk,* uma *framework* para quantificar a avaliação de risco de aplicações baseada no pedido de permissões Android, quer de aplicações benignas como de aplicações maliciosas. Com um conjunto de 27274 aplicações benignas de categorias diferentes e transferidas da Google Play Store e 1260 aplicações com *malware* transferidas da *Android Malware Genome Project* (Zhou & Jiang, 2012). Aplicando uma classificação no intervalo [0, 10] para o risco de permissões, concluíram que a maioria das aplicações benignas não apresentam permissões de risco superior a 2. Por outro lado, as aplicações que contêm *malware* (malignas) apresentam riscos mais elevados. Os resultados podem ser vistos na Figura *10***.**

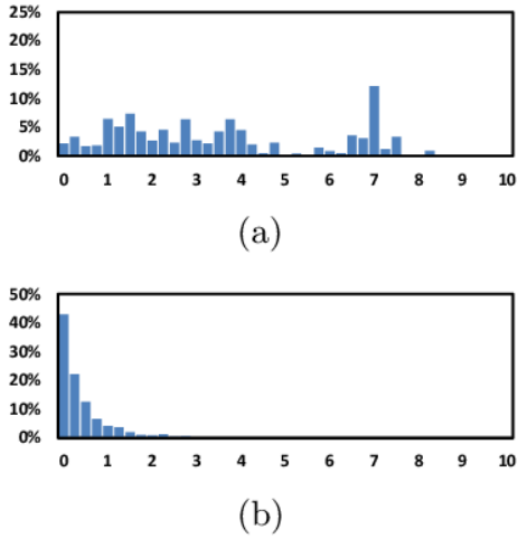

Figura 10 – Histogramas de nível de risco nas aplicações, (a) benignas, (b) malignas

Os autores deste trabalho também decidiram criar uma aplicação Android para ilustrar os resultados da *framework* DroidRisk, cujo nome da aplicação é o mesmo.

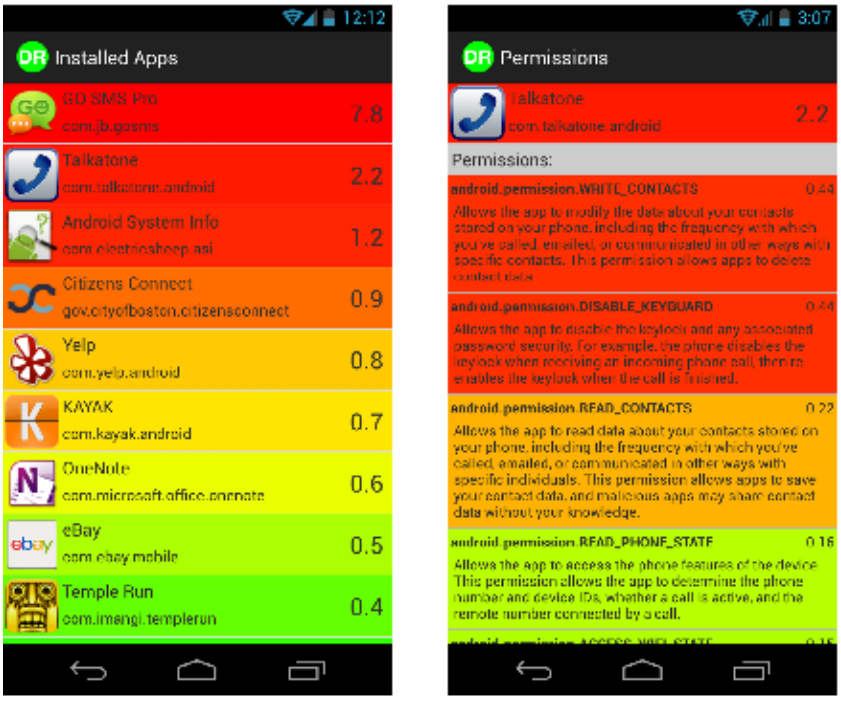

Figura 11 – Exemplo ilustrativo do DroidRisk, (a) uma lista de aplicações e seu respetivo nível de risco; (b) lista de permissões requeridas pela aplicação e respetivos níveis de risco *(Y. Wang et al., 2013)*

2.8 Conclusões

Após analisar e interiorizar os conceitos estudados na revisão da literatura, foi concluído que alguns destes conceitos iriam ser úteis para o desenvolvimento desta dissertação enquanto outros não teriam o mesmo efeito.

Uma vez que se pretende calcular um nível de segurança de aplicações Android durante o seu desenvolvimento, o foco vai mais para o código desenvolvido para essas aplicações com o objetivo de detetar falhas que possam vir a prejudicar o utilizador. Deste modo é preciso ter conhecimento das vulnerabilidades que existem para sistema Android, muitos CVE´s apresentam vulnerabilidades de aplicações deste sistema que provém de falhas no código. O *OWASP Mobile Security Project* tem muita informação sobre vulnerabilidades e sobre características especificas delas assim como explicações de como as vulnerabilidades podem ser exploradas, estes conceitos foram importantes para o desenvolvimento da solução.

Contudo, relativamente aos métodos de cálculo de risco, o método escolhido foi o CVSS, este método é o mais relevante porque quantifica as vulnerabilidades quanto às características a nível de exploração. Os restantes métodos apresentam uma variável que não é possível utilizar no contexto desta dissertação, o impacto no negócio. Para esta dissertação o objetivo é analisar mais que uma aplicação sendo que não é possível calcular o impacto no modelo de negócio de cada uma dessas aplicações.

Por fim, não foram encontrados trabalhos que cumprissem com o objetivo desta dissertação, apenas foi encontrado um trabalho que quantifica aplicações relativamente à sua segurança. Porém essa classificação é apenas baseada nas permissões que essas aplicações pedem. Isto não é o que se pretende para esta dissertação sendo que este trabalho não teve contribuição para o desenvolvimento da solução.

3 Desenvolvimento

Este capítulo foca-se na apresentação do sistema desenvolvido para resolver o problema de investigação e chegar aos objetivos definidos, referidos na secção 1.4. Começando pela apresentação da arquitetura do sistema e por explicar o funcionamento dos principais componentes em torno do cálculo do índice de risco de segurança em aplicações Android, assim como as tecnologias utilizadas pelo sistema. Por fim, é apresentada uma explicação mais detalhada sobre o cálculo do índice de risco com apresentação da fórmula aplicada.

3.1 Introdução

Esta dissertação está integrada num projeto de maior dimensão, designado por AppSentinel. O AppSentinel é um projeto de investigação realizado em conjunto pela Aptoide e pelo Iscte – Instituto Universitário de Lisboa que visa procurar resolver problemas de segurança em aplicações Android. O cálculo de risco de segurança de uma aplicação Android é uma das funcionalidades que o sistema final deverá oferecer como uma métrica extremamente importante para os gestores das lojas de aplicações (App Stores) poderem tomar decisões relativamente à aceitação ou rejeição das mesmas na loja. O projeto AppSentinel encontra-se dividido em duas grandes componentes, a deteção de padrões de *malware* e a análise de vulnerabilidades de segurança em aplicações Android. O tema desta dissertação está integrado na segunda componente, sendo que o cálculo automático do risco para aplicações móveis dependerá do módulo de deteção de vulnerabilidades.

A análise de vulnerabilidades feita em aplicações Android, resulta de metodologias de análise estática ao código gerado por ferramentas que aplicam técnicas de engenharia reversa. Uma vez que a analise é feita diretamente sobre um ficheiro APK, não existe acesso ao código fonte originalmente criado pelo programador.

Este sistema tem como objetivo dar informação necessária aos programadores sobre como mitigar as vulnerabilidades detetadas pelo sistema. Este tipo de informações presidem numa base de conhecimento que pode receber mais propostas de resolução ao longo do tempo. O sistema também apresenta uma API que permite enviar novos APKs para serem analisados e também pedir feedback do APK analisado, inclusive o valor de risco associado à aplicação. Por último, o AppSentinel apresenta um conjunto de componentes de *software* escritos na linguagem de programação Python, conjugados de forma a poderem executar uma análise de segurança aos APKs submetidos pela API. Este processo inclui, a gestão de APKs submetidos no sistema, a conjugação e integração das ferramentas de análise e deteção de vulnerabilidades, a organização e categorização das vulnerabilidades de acordo com o

modelo *OWASP Mobile TOP 10*, a atribuição de *feedback* aos programadores e por fim, a classificação de risco associado ao APK submetido.

3.2 Arquitetura do Sistema

O sistema está preparado para receber identificadores de APKs das respetivas lojas de aplicações e guardá-las em base de dados para quando o processamento de análise for iniciado, o sistema descarrega as aplicações das lojas. A Figura *12* ilustra o sistema completo.

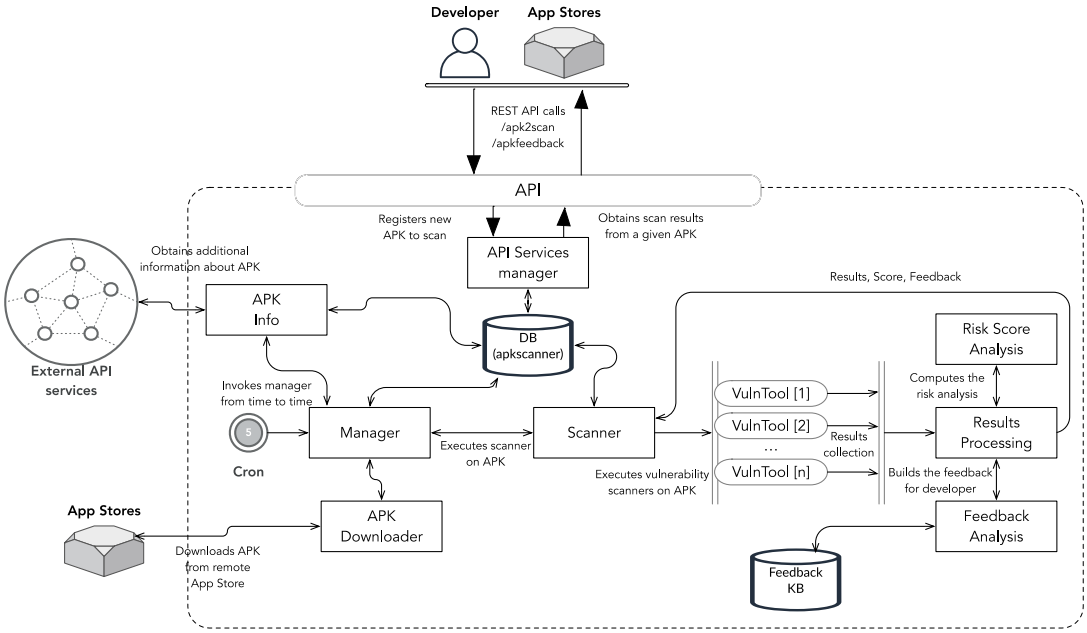

Figura 12 – Estrutura do servidor AppSentinel para o módulo de deteção e análise de vulnerabilidades(Projeto AppSentinel).

3.2.1 Armazenamento de informação

O sistema AppSentinel armazena toda a informação necessária para analisar um APK e todos os resultados gerados após uma análise executada também ao APK numa base de dados interna MySQL.

- Apk: contém toda a informação de um APK descarregado, por isso todos os meta dados existentes sobre o APK que está a ser analisado pelo sistema;
- Apk2scan: guarda a informação sobre o APK que foi submetido ao sistema, mas que ainda não foi analisado;
- Apkresults: guarda a informação dos resultados obtidos após uma análise executada a um dado APK;
- Apklevels: guarda a informação do índice de risco calculado para um dado APK.

3.2.2 API REST do Sistema

A API permite que sistemas externos possam interagir com operações internas do sistema de vulnerabilidades. Oferece funcionalidades para analisar APKs assim como para a coleção de resultados e *feedback* dos APKs analisados.

A entrada da API "apkscan" é usada por entidades externas para submeter um novo APK para análise. O cliente deve passar (através do método POST) um identificador de APK (identificador pode ser diferente consoante as lojas de aplicações) ou o URI para o ficheiro APK que se pretende analisar. A entrada opera da seguinte forma:

1. O cliente envia um pedido POST para o "apkscan", passando o identificador do APK ou o URI (identificador uniforme de recurso) do ficheiro APK;
2. O servidor recebe o pedido, verifica a existência do APK submetido
3. O servidor armazena o identificador do APK na base de dados interna
4. O servidor retorna uma mensagem com o resultado da operação. O resultado vem formatado em JSON.

Outra entrada da API, é o "apkfeedback". Esta entrada pode ser requerida por entidades externas através de um pedido GET passando o identificador do APK. O sistema retorna o resultado da análise do APK ao qual foi feito o pedido. A entrada opera da seguinte forma:

1. O cliente envia um pedido POST para o "apkfeedback", passando o identificador do APK
2. O servidor recebe o pedido, verifica a existência do APK submetido;
3. O servidor pede à base de dados o feedback e os resultados existentes para o APK em questão;
4. O servidor retorna uma mensagem com o resultado da operação. O resultado vem formatado em JSON.
    a. Caso um erro tenha ocorrido, é recebida uma mensagem com detalhes do erro
    b. Caso não ocorra erro e haja resultados para o APK dado, é enviado uma estrutura formatada em JSON com os seus resultados e feedback. A estrutura desta mensagem será apresentada numa das secções seguintes.

Por último, a entrada "vulnerabilities/levels". Esta entrada pode ser requerida através de um pedido GET passando o identificador do APK. O sistema retorna o valor de risco calculado para o APK que foi feito o pedido. A entrada opera da seguinte forma:

1. O cliente envia um pedido POST para o "vulnerabilities/levels", passando o identificador do APK
2. O servidor recebe o pedido, verifica a existência do APK submetido;
3. O servidor pede à base de dados o valor de risco correspondente ao APK em questão;

4. O servidor retorna uma mensagem com o resultado da operação. O resultado vem formatado em JSON.
    a. Caso um erro tenha ocorrido, é recebida uma mensagem com detalhes do erro
    b. Caso não ocorra erro e haja resultados para o APK dado, é enviado uma estrutura formatada em JSON com o valor de risco. Por exemplo: “{"status": "OK", "value": 0.58, “summary”: {(…)} }”

3.2.3 Deteção e análise de vulnerabilidades

A deteção e análise de vulnerabilidades, é todo o processo do sistema AppSentinel desde disponibilização de um novo APK ao sistema até à criação dos resultados derivados da análise. Isto inclui interação com a API, interação com o sistema de base de dados e interação com componentes *core* como o “manager” e o “scanner” que são responsáveis por gerir e inicializar, respetivamente, todo o processo de deteção e análise de vulnerabilidades.

O “manager” é responsável por verificar a existência de pedidos de análise e proceder a extração de toda a informação necessária para inicializar o processo de análise. E o “Scanner” é responsável por verificar a disponibilidade das ferramentas de análise de vulnerabilidades existentes e integradas no sistema. Foi desenvolvido um sistema de *Plugins* de modo a viabilizar a integração dessas ferramentas no sistema tornando o assim escalável e extensível. Na secção seguinte será explicado com mais detalhe a componente dos “*Plugins*”. Cada plugin analisa o mesmo APK e gera resultados únicos com as vulnerabilidades detetadas que serão agregados e concatenados de forma inteligente para evitar ao máximo duplicações ou mais relativamente a vulnerabilidades detetadas. O “Results processing” é o responsável por este processo e integra o “*OWASPEngine”* para organizar as vulnerabilidades consoante as suas características, este componente será explicado com mais detalhe na secção 3.2.5.

Por fim, após os resultados gerados pelas diferentes ferramentas estarem unificados num só conjunto de resultados, é processado o “Risk Score Analysis” que calcula o índice de risco associado à aplicação baseando-se nas vulnerabilidades que foram identificadas. O índice de risco é armazenado em base de dados e fica concluído o processo de cálculo de índice de risco de uma aplicação android até que seja novamente invocado o componente manager.

3.2.4 Plugins

Tal como pode ser visualizado na Figura *12*, podemos ver que existe uma integração de ferramentas de análise no projeto AppSentinel. Tendo em conta que estas ferramentas foram criadas por outros autores e que não foram propriamente implementadas com o propósito de integrar no AppSentinel, houve a necessidade de implementar *Plugins* de modo a contornar esse problema. Um *Plugin* é um

módulo de extensão e o seu objetivo é adicionar funções extra a um programa já desenvolvido. Cada ferramenta de análise tem um *plugin* associado e cada um deles pode ser ativado ou desativado sem afetar o funcionamento do sistema ou o comportamento das outras ferramentas. A metodologia de análise entre ferramentas é diferente entre as mesmas, cada ferramenta adota um comportamento diferente com características diferentes. Umas demoram mais tempo que outras a analisar APKs, os resultados apresentados também diferem entre ferramentas incluindo os próprios nomes dados às vulnerabilidades identificadas.

Para o cálculo do índice de risco de uma aplicação, as diferenças entre as ferramentas também vão criar um impacto que vai influenciar no valor atribuído. Uma vez que as ferramentas apresentam características diferentes, as vulnerabilidades detetadas por umas não podem ter o mesmo impacto que as vulnerabilidades detetadas por outra ferramenta mesmo quando são identificadas as mesmas vulnerabilidades. Consoante vários fatores, irão ter um peso associado na fórmula de cálculo. Esta característica de influência direta no cálculo de risco será explicada mais à frente na secção do cálculo do índice de risco.

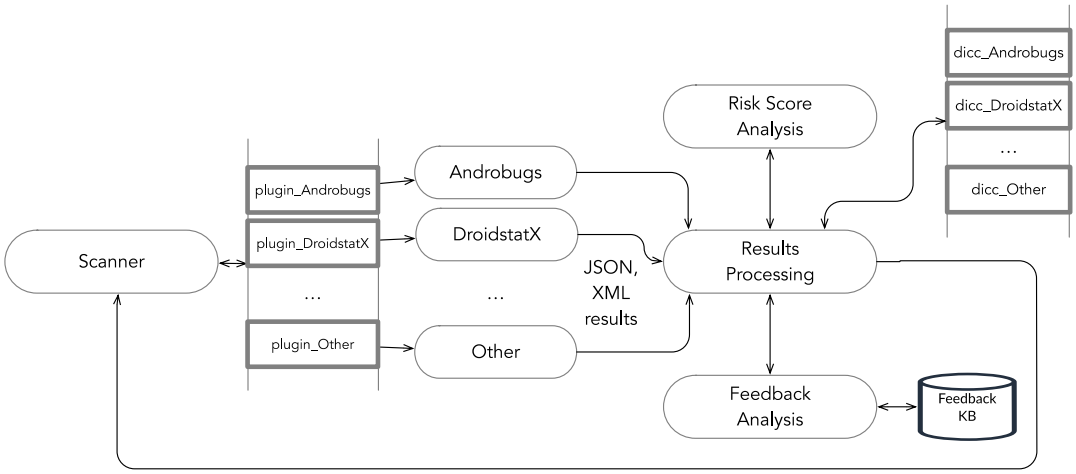

Figura 13 - Plugins no AppSentinel (Projeto AppSentinel).

Cada Plugin recebe um APK do scanner e em paralelo, cada ferramenta processa esse mesmo APK. Após o processamento, cada ferramenta gera o seu próprio relatório de análise em formato JSON. O relatório é um conjunto de vulnerabilidades identificadas constituídas por nome da vulnerabilidade, detalhes (explicação mais concreta sobre a vulnerabilidade, algumas ferramentas não disponibilizam, como por exemplo o DroidStatX), severidade e caso exista, links ou referencias sobre como mitigar a vulnerabilidade identificada. Como pode ser visualizado na Figura 14, para o mesmo APK, identificado pelo seu MD5, cada ferramenta gera o seu ficheiro de resultados. Neste caso as ferramentas SUPER (*Secure Unfied Powerful and Extensible Rust Android Analyzer*) Androbugs e DroidStatX.

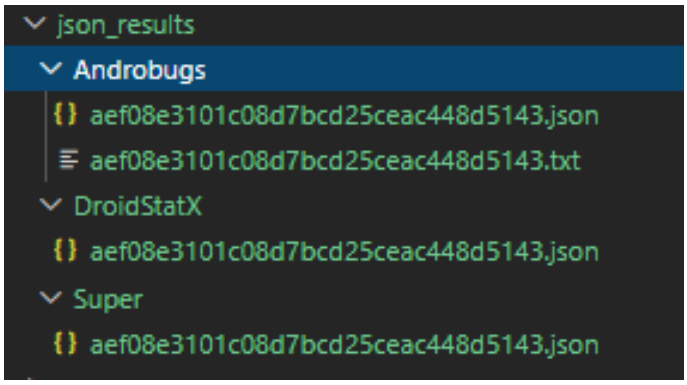

Figura 14 - exemplo da estrutura de ficheiros de resultados gerados por cada ferramenta para a mesma aplicação (identificada pelo md5) (Projeto AppsSentinel).

3.2.5 OWASP Engine

O *OWASPEngine* é o mecanismo que vai ser responsável por juntar e normalizar os resultados gerados pelas ferramentas assim como categorizar as vulnerabilidades de acordo com os níveis do *OWASP MOBILE TOP 10.* Esta componente, tem uma funcionalidade que compara as diferentes vulnerabilidades tentando encontrar semelhanças entre elas, usa essencialmente mecanismos de comparação de texto. Outra função do *OWASPEngine* passa por construir o relatório de resultados sobre o APK analisado assim como um *feedback* direcionado aos programadores, "*Feedback Analysis*". Esse *feedback* é constituído essencialmente por *links* e referencias para cada umas das vulnerabilidades detetadas, o objetivo é ajudar o programador a mitigar e a entender melhor as vulnerabilidades encontradas.

O relatório de resultados e o *feedback* a programadores gerados pelo *OWASPEngine* podem ser requeridos através da API disponibilizada pelo sistema. A resposta é uma mensagem em formato JSON composta por vários elementos "*status*", "*results history*" e "*results*". O primeiro elemento, "*status*", é um simples objeto JSON que contém um valor booleano indicando se o pedido foi bem sucedido ou não e em caso de erro uma mensagem que contem informação adicional.

O segundo elemento é o "*results history*" e é composto por um vetor de objetos JSON que representam o historial das diferentes análises feitas ao longo do tempo para uma determinada aplicação ou para diferentes versões da mesma. Este elemento é essencial porque permite auditar a maturidade da segurança de uma dada aplicação ao longo do tempo. O elemento "*results history*" apresenta a seguinte estrutura:

- **"*created at*":** representa a data e a hora da análise efetuada;
- **"*details*"**: uma *string* de dados que fornece detalhes extra sobre a atividade de análise. Estes detalhes podem incluir especificações das ferramentas ou especificações sobre quais as condições a analise foi realizada;

- **"*id*"**: cada atividade de análise é unicamente identificada pelo sistema. O identificador é parte do objeto JSON;
- **"*apkid*"**: representação única do APK da aplicação Android na loja de Aplicações. Este valor depende da loja de aplicações;
- **"*scantools*"**: representa a identificação das múltiplas ferramentas que foram usadas para conduzir a análise de vulnerabilidades desta execução de análise;
- **"*status*"**: representa o sucesso da análise que foi feita nesta instância

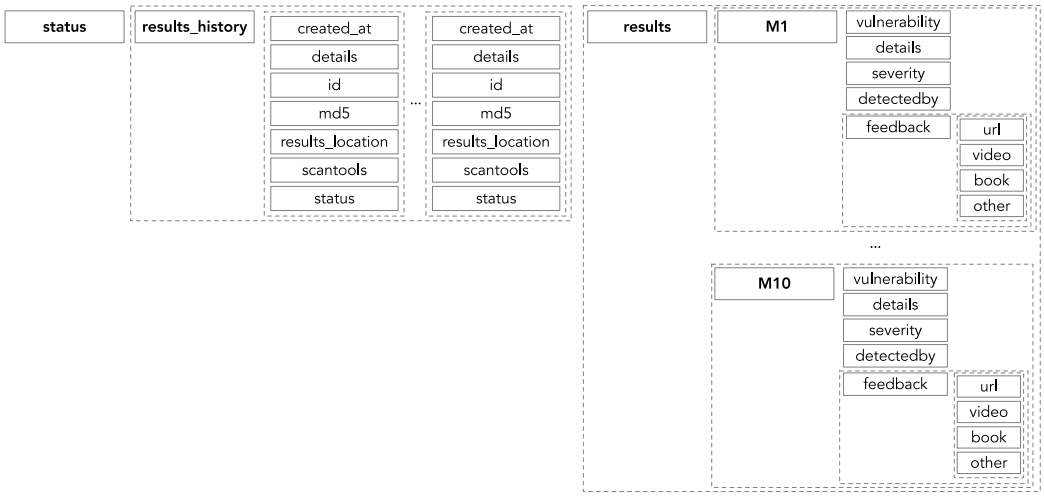

Figura 15 – estrutura da mensagem feedback em formato JSON (Projeto AppSentinel).

Por fim, o terceiro elemento, "*results*", é um conjunto de objetos JSON categorizados de acordo com nomenclatura do *OWASP Mobile TOP 10,* podendo agrupar as vulnerabilidades desde o "M1" até ao "M10". Em cada grupo, são aglomeradas as vulnerabilidades encontradas assim como as ferramentas que as detetaram e a severidade que foi atribuída pelas mesmas. Outra característica importante do *OWASPEngine* é a construção do campo "*feedback*", este recorre à *KnowledgeBase* para retirar todas os recursos de apoio associados à vulnerabilidade em questão. A abordagem tomada para este elemento ajuda a enumerar e a organizar as vulnerabilidades de acordo com as suas características e permite aos programadores perceberem rapidamente as questões de segurança sobre as suas aplicações Android. O elemento "*results*" apresenta a seguinte estrutura.

- **"*vulnerability*"**: frase única que descreve a vulnerabilidade muito sucintamente. Esta vulnerabilidade pode ser encontrada por mais que uma ferramenta de análise;
- **"*details*":** neste campo é apresentado uma descrição mais detalhada sobre a vulnerabilidade, em alguns casos com proposta de mitigação da mesma e como enumeração de CVE ou CWE;
- **"*severity*":** nem todas as vulnerabilidades detetadas têm a mesma classificação em termos de severidade e por isso este campo tem como objetivo agrupá-las de acordo com o seu nível de

risco. Quanto maior o nível de severidade mais critica é a vulnerabilidade, e podem variar entre vulnerabilidades altamente exploráveis que podem comprometer o utilizador e vulnerabilidades menos exploráveis que apenas podem dar informações de menor risco ao atacante, como por exemplo informações do ambiente onde a aplicação está a ser executada;

- **"*detectedby*":** neste campo é apresentado quais as ferramentas que detetaram a vulnerabilidade. É representado por um vetor onde contém o nome das ferramentas que podem variar entre apenas um nome, caso só uma ferramenta tenha detetado a vulnerabilidade, ou enumerando todas as ferramentas que tenham detetado essa mesma vulnerabilidade;
- **"*feedback*":** Este campo fornece um conjunto de informações sobre a vulnerabilidade, assim como informações mais detalhadas, problemas relacionados com o desenvolvimento, de que maneiras a vulnerabilidade pode afetar os utilizadores e eventuais formas de mitigação e correção da vulnerabilidade. De modo a educar os programadores de aplicações mobilem são disponibilizadas várias informações adicionais com o seguinte formato:
    - **"*url*":** lista de URLs que apontam para recursos online contento informação relevante sobre a vulnerabilidade;
    - **"*vídeo*":** ligações para vídeos que contém informações da vulnerabilidade, como explorá-la ou como corrigi-la;
    - **"*book*":** indicação de livros que referem a vulnerabilidade;
    - **"*other*":** outras informações relevantes sobre a vulnerabilidade.

O relatório de resultados é guardado no sistema de ficheiros do servidor do AppSentinel. A calculadora é um dos recursos do sistema que depende do relatório de análise e *feedback* para calcular o índice de risco de uma determinada aplicação. Esses relatórios são guardados numa diretoria própria onde todos os relatórios de cada APK apresentam apenas a estrutura do elemento "*results*". A calculadora investiga quais os tipos de vulnerabilidades que foram detetadas iterando o relatório por cada categoria do OWASP e tendo em especial atenção aos campos "*vulnerability*" e "*detectedby*". Sabendo qual o nome da vulnerabilidade, sabendo por qual ferramenta foi detetada e sabendo qual a sua categoria do OWASP, a calculadora faz uma pesquisa rápida ao dicionário da respetiva ferramenta e retira as palavras chave associadas à vulnerabilidade. Com essas palavras chave, torna-se possível saber qual o valor de risco atribuído à vulnerabilidade em questão através das informações guardadas na *KnowledgeBase*. Repetindo este processo para todas as vulnerabilidades encontradas no relatório, a calculadora irá guardá-las todas e posteriormente utilizá-las para o cálculo final do índice de risco.

3.2.6 KnowledgeBase e Dicionários

Conforme foi referido anteriormente no sumário da estrutura do sistema, na seção 3.2, as bibliotecas *KnowledgeBase* e Dicionários são peças fundamentais para o funcionamento do sistema e também têm grande peso para a qualidade do mesmo.

Sempre que um programador quiser integrar uma nova ferramenta de análise de vulnerabilidades, para além de implementar o *plugin* deve também fornecer um dicionário específico. Este dicionário deve conter todos os resultados possíveis que a ferramenta pode retornar num formato generalizado e integrado com o sistema. Outra característica é que os resultados presentes num dicionário devem estar categorizados de acordo com o *OWASP Mobile Top 10*, uma vantagem para quando as ferramentas não têm este tipo de categorização. Caso já tenham de raiz, o critério de categorização não deve ser alterado. Os dicionários são utilizados principalmente porque a maioria das ferramentas não vêm categorizadas de acordo com o *OWASP Mobile Top 10,* mas também foram um contributo essencial para calcular a severidade das vulnerabilidades de cada ferramenta. O facto de estas estarem todas agrupadas num ficheiro apenas e já categorizadas de acordo com as suas características, ajudou no processo de análise e compreensão das mesmas. Foi um passo importante para o cálculo final do índice de risco para aplicações.

Contudo, para o funcionamento da calculadora em termos de código, os dicionários também são uma peça importante porque ao aceder a esses recursos a calculadora vai encontrar as características necessárias da vulnerabilidade para atribuir o nível de risco. Essas características são palavras chave que definem o tipo de vulnerabilidade. A atribuição de palavras chave é feita da mesma forma para todos os dicionários e o objetivo é poder uniformizar as vulnerabilidades idênticas entre as diferentes ferramentas. Isto porque ferramentas diferentes identificam as mesmas vulnerabilidades, mas descrevem-nas com palavras e frases diferentes, mas com o mesmo significado, inclusive o próprio nome da vulnerabilidade. Os dicionários apresentam a seguinte estrutura:

- **“name”:** nome da vulnerabilidade que a ferramenta atribui;
- **“category”:** Categoria do *OWASP Mobile TOP 10* correspondente;
- **“level”:** Nível de severidade da vulnerabilidade atribuído pela própria ferramenta, este não é o nível utilizado para o cálculo final do índice de risco para a aplicação.
- **“Keywords”:** Vetor agregador de palavras chave relevantes para poder relacionar com as vulnerabilidades identificadas por outras ferramentas.

Em baixo, na Figura 16, pode ser observado um caso em que duas ferramentas apresentam nomes diferentes para o mesmo tipo de vulnerabilidade.

Figura 16 - Comparação entre a mesma vulnerabilidade detetada por ferramentas diferentes (exemplos retirados de dicionários, em cima SUPER, em baixo DroidStatX) (Projeto AppsSentinel).

O *KnowledgeBase* é uma base de dados de conhecimento que guarda um vasto conjunto de informações sobre as vulnerabilidades. Esta base de dados é fundamental para o cálculo de risco e para a criação dos comentários com conteúdos didáticos que deverão ser enviados para o programador, incluindo um nível de severidade mais preciso e detalhado para cada vulnerabilidade. O *KnowledgeBase* também está organizado por categorias de acordo com o *OWASP Mobile TOP 10* e apresenta a seguinte estrutura:

- **"category":** Categoria OWASP Mobile TOP 10 para a qual serão enquadradas as informações dos restantes campos;
- **"links_by_category":** ligações com informações relevantes sobre vulnerabilidades da categoria correspondente;
- **"books_by_category":** livros sobre vulnerabilidades da categoria correspondente;
- **"articles_by_category":** artigos com informações relevantes sobre vulnerabilidades da categoria correspondente;
- **"keywords":** Este campo representa um vetor onde são guardadas as informações sobre uma determinada vulnerabilidade identificada por uma ou mais palavras chave;
    - **"name":** Vetor onde são guardadas as palavras chave de uma vulnerabilidade, poderá conter mais do que uma palavra;
    - **"score":** Valor numérico (float) correspondente ao nível de severidade da vulnerabilidade. Estes valores variam ente zero e dez, quanto maior este valor maior a severidade da vulnerabilidade. Na maioria dos casos o valor pode tratar-se de um CVSS que pode ser da versão dois (CVSSv2) ou da versão três (CVSSv3)
    - **"vectorString":** Caso o campo "score" for um CVSS, aqui será atribuído um *vetor string,* que é uma representação textual das métricas usadas para determinar o nível;

- **"links":** ligações com informações relevantes sobre a vulnerabilidade, incluindo uma proposta de mitigação da mesma caso exista;
- **"books":** livros que possam referenciar a vulnerabilidade, conter mais informações de como explorá-la ou mitigá-la;
- **"articles**": artigos com informações relevantes sobre vulnerabilidade;


```
{
    "name": ["xss"] ,
    "score": 5.4,
    "vectorString": "AV:N/AC:M/Au:N/C:C/I:C/A:C",
    "links": ["https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)"],
    "books": [],
    "articles": []
},
```


Figura 17 – Exemplo de um tipo de vulnerabilidade (vulnerabilidades sujeitas a ataques XSS) na *KnowledgeBase*, Neste exemplo o score e o vectorString são CVSSv2. (Projeto AppsSentinel)

3.2.7 Calculadora de risco do APK

O cálculo do índice de risco é uma funcionalidade fundamental do sistema AppSentinel e é o foco principal desta dissertação. No entanto, todo o processo descrito anteriormente foi necessário desenvolver para possibilitar o cálculo do risco, esta classificação associada ao risco de código desenvolvido não seria possível sem primeiro fazer uma identificação das vulnerabilidades presentes num APK. O processo do cálculo de risco durante a análise de um APK inicia-se principalmente após as ferramentas de análise terminarem a execução e enviarem os resultados para o módulo responsável pelo processamento dos mesmos. Esta secção tem o intuito de explicar o processo de cálculo a partir desse momento para dar mais enfâse ao objetivo desta dissertação.

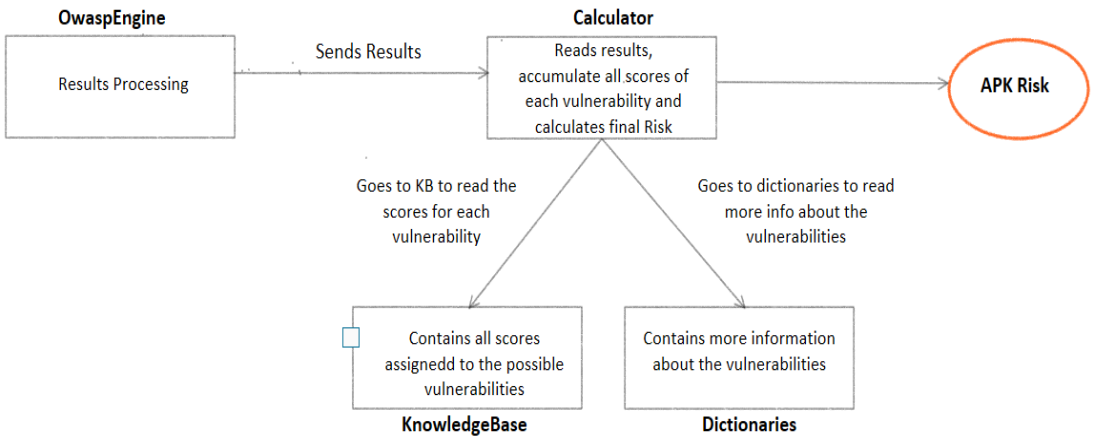

Figura 18 – Estrutura da componente de cálculo do índice de risco

A Figura *18* visa a ilustrar de forma sucinta o funcionamento da componente de cálculo do índice de risco para as aplicações submetidas ao AppSentinel. Para que seja possível calcular esse índice, o

módulo C*alculator* depende dos resultados tratados pelo *OWASPEngine,* este que é responsável por normalizar os resultados provenientes das diferentes ferramentas de análise assim como por organizar esses resultados agrupando as vulnerabilidades pelas suas respetivas categorias. Tendo os resultados processados, a calculadora recebe-os e processa as vulnerabilidades registadas uma a uma. O processo de análise individual de uma vulnerabilidade é o seguinte:

1. *Calculator* começa por identificar qual a categoria OWASP e de seguida itera sobre as vulnerabilidades dentro dessa categoria;
2. Guarda a categoria que está a analisar e verifica o tipo de vulnerabilidade identificada pelo campo "vulnerability" e por quais ferramentas foram detetadas através do campo "detectedby", também guarda essas informações em memória;
3. Tendo informação da categoria OWASP (respetivo M), a informação da vulnerabilidade e por qual ferramenta foi detetada (pode ser detetada por mais que uma ferramenta), a calculadora consulta o dicionário onde se encontra a vulnerabilidade detetada e extrai as palavras chave atribuídas a essa vulnerabilidade;
4. Obtendo as palavras chave e ainda sabendo a categoria da vulnerabilidade, a calculadora consulta o K*nowledgeBase* e procura na respetiva categoria as palavras chave que identificam aquela vulnerabilidade, ao encontrá-las, a calculadora tem acesso ao valor numérico que nivela o risco da vulnerabilidade. Caso for um CVSS, também lê o campo "*vectorString"* mesmo sabendo que este não é necessário para o cálculo do índice de risco da aplicação.
5. A calculadora guarda o nível de risco associado a vulnerabilidade num dicionário Python onde ira juntar todos os valores de risco das vulnerabilidades para posteriormente aplicá-los na fórmula de calculo para o índice de risco. A fórmula será apresentada mais à frente na secção 3.4.1.
6. O processo de atribuição de risco a uma vulnerabilidade detetada é finalizado e caso haja mais vulnerabilidades por analisar, o processo é repetido até que todo o relatório de resultados seja totalmente analisado.

Grande parte do processamento de cálculo do índice risco passa por quantificar o nível de segurança das vulnerabilidades detetadas, porém também é preciso olhar para as ferramentas de análise de vulnerabilidades e ver os respetivos impactos no cálculo do índice de risco. Uma vez que as ferramentas apresentam características diferentes, a cada uma delas é atribuído um valor que representa o seu peso para a fórmula de calculo. Esse peso é baseado num conjunto de fatores que serão explicados na secção 3.4.2.

Com base na Figura *12*, é possível ver que um dos últimos componentes do sistema é o *"Risk Score Analysis",* que representa o *Calculator* na Figura *18.* Este componente é responsável por efetuar todo

o processo de cálculo de risco para uma aplicação Android. Esta componente é uma classe Python e é invocada pelo *OWASPEngine*.

O Calculator quando é chamado, é lhe atribuído um identificador de APK, por exemplo um md5. Quando inicia o construtor, a classe guarda de imediato como variável global esse md5, para mais tarde ler o seu respetivo output final. Também, ainda no construtor, é chamada a função *setup()* para retirar informação sobre quais os plugins disponíveis e seus respetivos valores de ponderação. A função *calculate()*, apenas pode ser chamada por uma instância proveniente de fora da classe. Uma vez chamada essa função, é iniciado o processo de cálculo cujo fluxo de processamento até ao cálculo do risco, pode ser visto na Figura *19*.

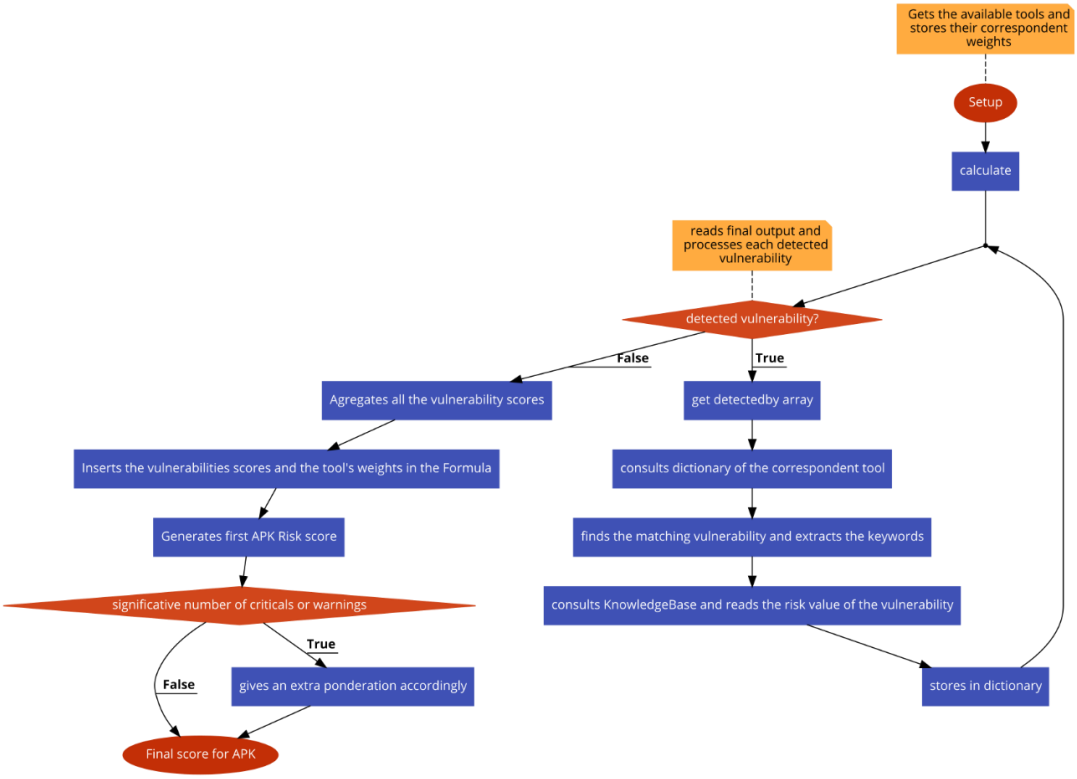

Figura 19 – Fluxograma do componente Calculator (fluxograma gerado por *Code2Flow*)

Tal como já tinha sido referido, o processo de cálculo depende da leitura do *output* final gerado para ler as vulnerabilidades que foram detetadas. Para cada vulnerabilidade detetada, o sistema consulta o *KnowledgeBase* para atribuir o score correspondente à vulnerabilidade em questão. Caso a vulnerabilidade seja detetada por mais que uma ferramenta, o mesmo processo é feito para cada ferramenta. Só depois de todas as vulnerabilidades detetadas tiverem um score atribuído, é que o sistema passa à fase de cálculo para a obtenção de um valor de risco para a aplicação analisada. Caso haja um número significativo de vulnerabilidades de severidade alta, é adicionada uma ponderação extra ao risco final.

Em conclusão, de acordo com este método, calcular automaticamente um valor de risco de segurança associado ao código implementado para uma aplicação móvel, torna-se possível com a ajuda de ferramentas de análise estática para deteção de vulnerabilidades.

3.3 Tecnologias utilizadas

O sistema AppSentinel foi desenvolvido com a linguagem de programação *Python*, sendo utilizadas as duas versões (python3.6 e python2.7) e para funcionar em ambiente Linux. A razão pela qual foi feita a escolha da linguagem, baseia-se no facto de a maioria das ferramentas de análise de vulnerabilidades serem desenvolvidas em *Python* também. Assim a decisão foi feita de modo a haver mais uniformidade com as ferramentas.

Python é uma linguagem criada em 1991 por Guido van Rossum, um programador holandês que buscava, entre outros objetivos, por simplicidade de código, o que torna *Python* uma linguagem que necessita de menos linhas de código para trazer os resultados esperados, o que não a torna menos poderosa (Foundation, 2016). Os componentes server, plugins e manager foram criados em *Python*, e por ter divergência de código entre versões desta linguagem, com exceção do server estes componentes foram criados em duas versões (python 3.6 e python 2.7).

Para a implementação da API disponível no sistema, foi utilizada a *framework Flask*. O *Flask* é um *Web Server Gateway Interface* (WSGI) e apresenta a característica de não consumir muitos recursos. Foi desenhado para ser fácil de usar em fases iniciais de desenvolvimento com escalabilidade para implementar aplicações mais complexas. O *Flask* tornou-se uma das *frameworks* para Web mais populares do *Python*. A *framework* oferece sugestões, mas não apresenta nenhuma estrutura de projeto que deve ser seguida assim como outras dependências. Os programadores são livres de escolher as ferramentas e bibliotecas que quiserem (Flask, 2020).

Para a implementação da interface gráfica *web*, foi usada a ferramenta Angular. Angular é uma ferramenta para desenvolvimento de aplicações *web*, baseado em *TypeScript* e de código fonte aberto. Atualmente é mantida por uma equipa da Google e pela comunidade.

De modo a facilitar a instalação do sistema, foi feito um ficheiro de configuração para Docker. O Docker começou como um projeto de código fonte aberto pela dotCloud em 2013. Rapidamente ganhou popularidade e o seu crescimento deu origem à Docker Inc. O projeto apresenta o conceito de virtualização por meio de contentor (*Docker container*) que virtualizam ao nível do sistema operativo, ao contrário das máquinas virtuais que o fazem ao nível do hardware usando parte dos recursos computacionais da máquina anfitriã (Merkel, 2014).

3.4 Cálculo do índice de Risco de Segurança da Aplicação

Um dos principais objetivos deste trabalho é o cálculo do Índice de Risco de Segurança da Aplicação (IRSA) e a ameaça representada pelo desenvolvimento de aplicações Android. Esta seção centra-se na explicação do mecanismo de cálculo. Porém toda a explicação do funcionamento do sistema em geral é importante para entender o mecanismo de cálculo, uma vez que existe dependência de todo o processo anteriormente descrito.

3.4.1 Índice de Risco de Segurança da Aplicação

De modo a chegar a um valor único que determina o risco associado a um APK, foi definida uma fórmula de média aritmética ponderada. A média aritmética ponderada, ao contrário da média aritmética comum, associa um peso a cada termo, existem termos que contribuem mais que outros. Tendo em conta que existem vulnerabilidades com pesos diferentes e ferramentas de qualidade diferente, assumimos ponderar tanto as vulnerabilidades como as ferramentas utilizadas. Nem todas as ferramentas podem contribuir de forma igual, existem um conjunto de fatores que devem ser tomados em conta na ponderação das ferramentas, em termos qualitativos, existem umas com peso maior do que outras.

A fórmula de calculo utilizada para calcular a média ponderada por cada ferramenta é apresentada em baixo:

$$Risk_t = \frac{\sum_{v=1}^{\infty} Critical_v \; x \; P_c + Warning_v \; x \; P_w + Notice_v \; x \; P_n}{v} \qquad (1)$$

Cada ferramenta deteta v vulnerabilidades para um APK. Essas vulnerabilidades são divididas de acordo com o seu grau de severidade e multiplicadas pelo peso associado a essa severidade. Para as vulnerabilidades críticas é multiplicada uma ponderação P_c normalmente maior que as restantes. Para os *warnings* uma ponderação mais intermédia P_w e para as vulnerabilidades do tipo *notice,* uma ponderação mais baixa que as restantes P_n. Por fim, é feito o somatório entre cada grupo de severidade e depois divido pelo número total de vulnerabilidades, v. Caso seja apenas utilizada uma ferramenta ou caso outras ferramentas falhem a análise, restando apenas uma, esta será a fórmula final do cálculo de risco da aplicação, $Risk_t$.

Outro aspeto importante deste cálculo cai sobre as ponderações das vulnerabilidades, é preciso ter em conta que dentro do escopo de vulnerabilidades críticas, por exemplo, existem umas mais críticas

que outras. Deste modo não podemos assumir uma ponderação igual para todas as vulnerabilidades desse grupo de severidade. O mesmo acontece para o grupo dos *warnings* e dos *notices.* Então temos o seguinte para cada grupo de severidade:

$$Severity_v \; x \; P_s = (Severity_{v1} \; x \; P_{s1} + Severity_{v2} \; x \; P_{s2} + \cdots + Severity_{vn} \; x \; P_{sn}) \quad (2)$$

Onde $Severity_{vn}$ corresponde ao número de vulnerabilidades daquele grupo de severidade para uma ponderação específica correspondente, P_{sn} .

Tendo em conta que as ponderações caiem sobre as vulnerabilidades e sobre as ferramentas, a média das vulnerabilidades é feita individualmente por cada ferramenta. Só depois é feito a média final das médias geradas por cada ferramenta atribuindo-lhes o peso respetivo de cada uma. Analisando primeiro

No entanto, é preciso ter em conta que o cálculo do risco poderá depender de mais do que uma ferramenta e que cada uma delas contribui com um peso diferente. Depois de cada ferramenta calcular o seu risco individual, será calculada uma nova média ponderada para o número total de ferramentas usadas na análise. Ou seja:

$$Risk = \frac{Risk_{t1} \; x \; P_{t1} + Risk_{t2} \; x \; P_{t2} + \cdots + Risk_{tn} \; x \; P_{tn}}{t_{total}} \quad (3)$$

Onde t_{total} corresponde ao número total de ferramentas utilizadas na análise de um APK.
Por fim ficamos com a seguinte fórmula final:

$$Risk = \frac{\sum_{t=1}^{\infty} (\frac{\sum_{v=1}^{\infty} Critical_v \; x \; P_c + Warning_v \; x \; P_w + Notice_v \; x \; P_n}{v}) \; x \; P_t}{t} \quad (4)$$

- P_c – Ponderação de vulnerabilidade critica
- P_w – Ponderação de vulnerabilidade de aviso
- P_n – Ponderação de vulnerabilidade de notificação
- P_t – Ponderação por ferramenta
- v – Número total de vulnerabilidades encontradas
- t – Número total de ferramentas

Após a obtenção do índice de risco através da fórmula sistema, ainda existe um processo de verificação da quantidade de vulnerabilidades do tipo *critical* e do tipo *warning*. Foi decidido que os APK's que apresentassem números significativos desse tipo de vulnerabilidades deveriam receber um acréscimo no valor de risco. O objetivo é agravar o nível de segurança quando há deteção de muitos casos que podem pôr em risco o utilizador. O limite mínimo atribuído para adicionar as ponderações extra foi de dez vulnerabilidades por tipo. Por outras palavras, se houver dez ou mais vulnerabilidades do tipo *warning* ou *critical*, é adicionado um peso extra ao valor calculado pela fórmula final.

$$final\ risk\ (x) = final\ risk + x + y \tag{5}$$

$$x = \begin{cases} 0, & criticals < 0 \\ 0.01 * critical, & criticals \geq 10 \end{cases} \tag{6}$$

$$y = \begin{cases} 0, & warnings < 0 \\ 0.002 * warnings, & warnings \geq 10 \end{cases} \tag{7}$$

3.4.2 Ponderações das Ferramentas

Calcular as ponderações para cada ferramenta não é um processo exato e não existe uma fórmula matemática para tal. A soma total das ponderações das ferramentas tem de ser igual ao número total de ferramentas utilizadas na análise. Sabendo que as ferramentas têm pesos diferentes, tal como foi dito anteriormente, é preciso fazer uma análise comparativa a cada uma delas de modo a chegar às conclusões certas. Para isso, foram definidas algumas características como, referências científicas, feedback de utilizadores baseado na pontuação do repositório GitHub e último suporte dado à ferramenta baseado no último *commit* também no repositório do GitHub de cada ferramenta. Caso exista um estudo científico que compare as ferramentas entre as quais se pretende utilizar no sistema, então as ponderações devem se reger pelo mesmo estudo. Caso não exista, a ordem definida para atribuição de ponderações utilizadas nesta dissertação foi a seguinte:

- Com maior peso na ponderação, primeiro vem a existência de estudos científicos sobre as ferramentas. Quanto mais referencias as ferramentas tiverem, sendo que essas referências apontem para boa qualidade da ferramenta, maior serão as suas ponderações. Por exemplo:
    - O AndroBugs é referenciado em vários estudos científicos, (Li et al., 2017)foi apresentado no *BlackHat Europe 2015* e também já foi referenciado mais do que uma vez no *Hall of Fame* de várias grandes empresas como Google, Facebook, Twitter, entre outras.

- Para o Qark, (Kulkarni & Javaid, 2018) fizeram um estudo onde comparavam a eficácia entre várias ferramentas de análise de vulnerabilidades *open-source* e concluíram que o Qark obteve os melhores resultados.

- De seguida, e com menos peso vem a popularidade do repositório na comunidade GitHub baseado na quantidade de estrelas que o repositório tem. Quanto maior esse número, maior será a sua popularidade.
- Por último e também importante, é tido em conta se as ferramentas continuam a receber suporte ou há quanto tempo não recebem, verificando o último *commit* nos seus repositórios GitHub.

Ferramentas	Ref. Científicas	Estrelas GitHub	Último Suporte
Androbugs	Sim	788	5 anos
DroidStatX	Não	18	10 meses
Super	Não	312	2 anos
Qark	Sim	2300	16 meses

Tabela 1 – Comparação entre ferramentas para determinar as ponderações de cada uma para o cálculo do risco

Na Tabela *1*, é possível ver os resultados da análise descrita a quatro ferramentas distintas, Androbugs, DroidStatX, Super e Qark. Rapidamente vemos que o Qark e o Androbugs terão as ponderações mais altas, no entanto o Qark apresenta maior popularidade e suporte mais recente, logo seria o mais alto dos quatro. Entre o DroidStatX e o Super, o Super apresenta maior popularidade logo o seu peso será maior em comparação com o DroidStatX. Teríamos as seguintes ponderações para cada ferramenta: Qark 1.55; Androbugs 1.20; Super 0.80; DroidStatX: 0.45.

3.4.3 Ponderações das Vulnerabilidades

Tal como já foi dito anteriormente em 3.4.1, nem todas as vulnerabilidades pertencentes ao mesmo grupo de severidade têm a mesma ponderação. Dentro desse mesmo escopo de severidade existem sempre diferenças, umas podem ser mais facilmente exploráveis que outras ou umas podem afetar mais o utilizador que outras. Para diferenciar melhor essas vulnerabilidades, foi utilizado o CVSS para atribuir valores numéricos a cada vulnerabilidade quando possível. Sabendo que o CVSS captura as principais características técnicas das vulnerabilidades, explicado em 2.5.3, foi feita uma pesquisa individual para cada vulnerabilidade Android que possa ser detetada pelas ferramentas integradas no sistema. Em alguns casos as pontuações CVSS já são atribuídas por haver vulnerabilidades publicadas

como CVE´s, para outros casos, teve de ser feita uma pesquisa mais detalhada sobre as características da vulnerabilidade. De modo a ser possível calcular um CVSS, é necessário saber as características de uma vulnerabilidade para concluir que tipo de ataque pode ser executado para explorá-la. Para este estudo apenas foram utilizadas as métricas-base do CVSS.

O processo foi bastante complexo ao início e um pouco complicado, foi necessário analisar várias vulnerabilidades das ferramentas e seus detalhes uma a uma de modo a atribuir um CVSS. Na plataforma Android o nível e a facilidade de exploração de vulnerabilidades variam consoante a versão do dispositivo. Por exemplo, versões inferiores ao Android *KitKat* 4.4 apresentam muito mais vulnerabilidades do que as versões superiores. Por isso houve um esforço para calcular os valores de risco para versões iguais ou superiores à versão referida anteriormente, tendo em conta que atualmente a maioria dos utilizadores Android têm os dispositivos atualizados.

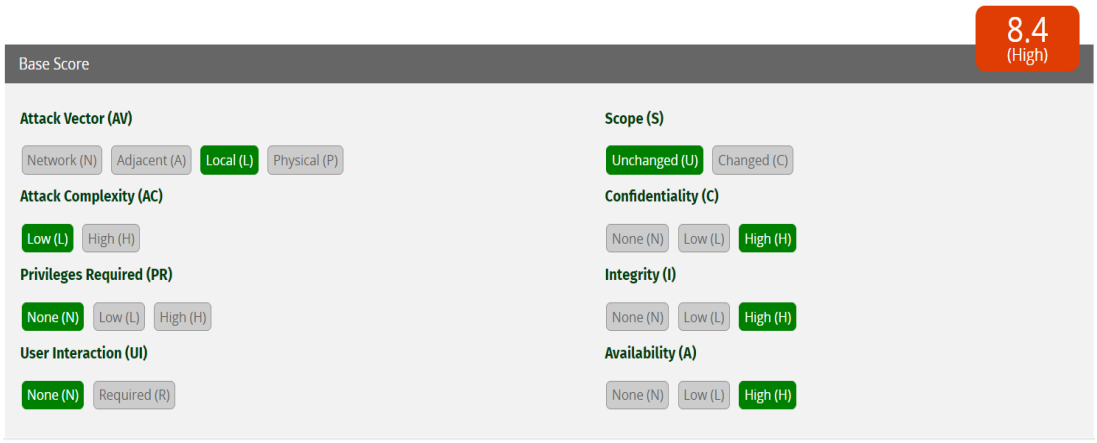

Figura 20 – Calculadora oficial da FIRST para o cálculo do "*base score*" do CVSS atribuindo as características de um possível ataque (Common Vulnerability Scoring System Version 3.1 Calculator, 2020)

Depois foi feita uma comparação entre ferramentas e em muitos casos, era possível atribuir o mesmo score a outras vulnerabilidades de outras ferramentas, sendo que o cálculo foi feito apenas para uma. Idêntico ao problema de normalização dos resultados, vulnerabilidades iguais de ferramentas diferentes, devem ter o mesmo nível de severidade CVSS e em alguns casos, vulnerabilidades diferentes também podem ter os mesmos valores de risco tendo em conta que sofrem o mesmo tipo de ataque.

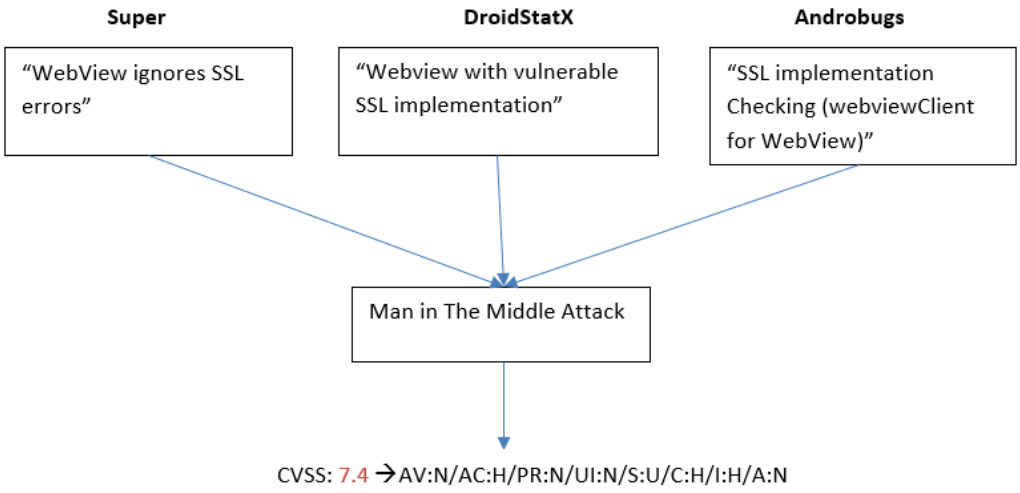

Figura 21 – Exemplo de atribuição de um CVSS para vulnerabilidades que sofrem o mesmo tipo de ataque.

3.5 Interação com utilizador

Esta secção tem um intuito de mostrar uma aplicação funcional e amigável (*user friendly*) ao utilizador. Para isso foi desenvolvida uma interface gráfica *web* com a ferramenta Angular. A aplicação *web* comunica com o servidor através da API desenvolvida e o processo inicial começa com a atribuição do APK ao sistema, invocando o ponto de acesso indicado ("apkscan").

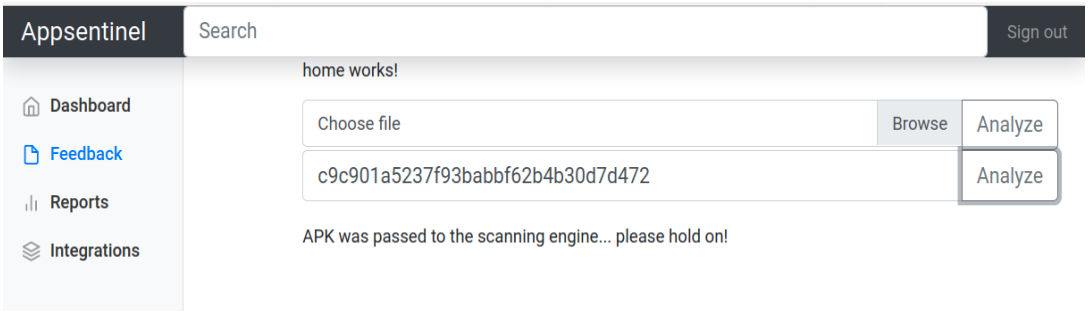

Figura 22 – Registo de APK para análise no AppSentinel.

Após o utilizador submeter a aplicação para análise, este recebe uma mensagem indicando se o APK já tinha sido analisado anteriormente, caso contrário, este indicará que foi aceite e a análise será efetuada nos próximos instantes. A análise é feita do lado do *back-end* e o tempo pode variar consoante as características de processamento do servidor. Deste modo, o utilizador fica à espera que a análise termine. Quando terminar, o utilizador pode ir à lista de APKs analisados e inspecionar a sua aplicação para visualizar os resultados gerados pelo AppSentinel.

Ao inspecionar, o utilizador é redirecionado para uma página que apresenta estatísticas relativamente ao tipo de vulnerabilidades detetadas assim como o valor do índice de risco de segurança junto do nome da aplicação.

Para obter mais informações sobre o índice de risco de segurança da aplicação, o utilizador pode clicar sobre o valor no ecrã e é aberto um "*modal*" com uma explicação mais detalhada sobre como este valor é obtido e o que ele significa. A justificação do valor obtido é importante para ajudar programadores e gestores de controlo de qualidade a interpretar as características que levaram à atribuição daquele nível de risco para poderem justificar a decisão de atualizarem o código da aplicação ou não. Nesta funcionalidade é dado uma sugestão de como devem abordar a aplicação consoante o valor do índice de risco calculado.

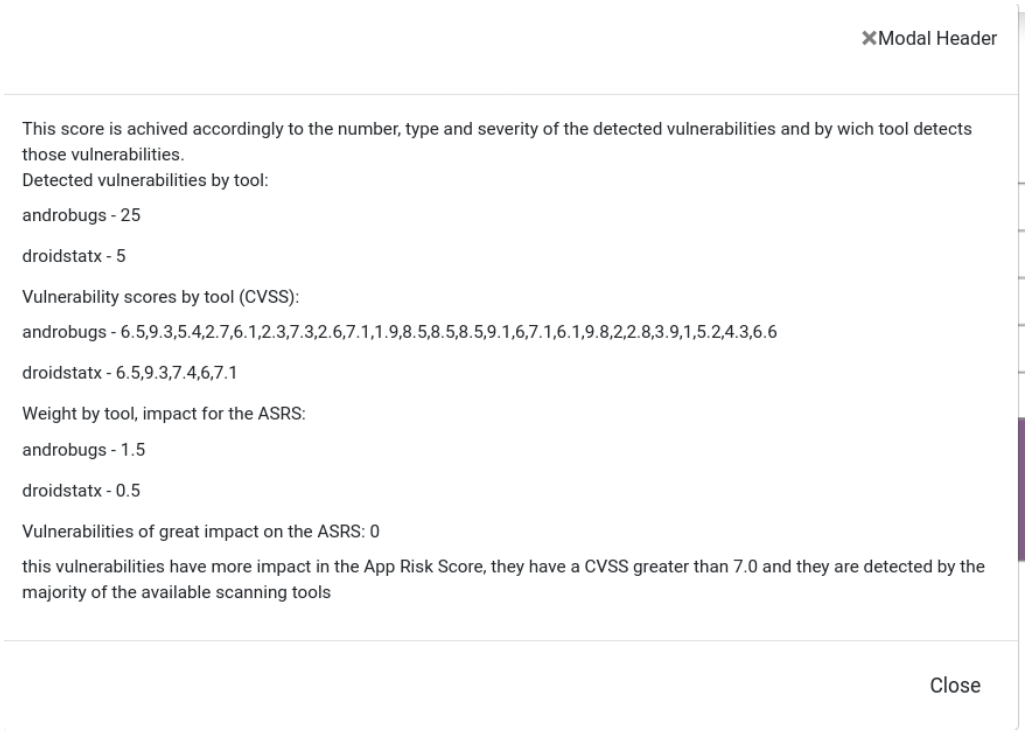

Figura 23 - Explicação da obtenção do índice de risco de segurança

A explicação apresentada ao utilizador é organizada de forma a enumerar certas características separadas por cada ferramenta de análise. O conjunto de características inclui, o número de vulnerabilidades detetados por cada ferramenta, os CVSSs de cada vulnerabilidade, o peso atribuído a cada ferramenta durante o cálculo e ainda o número de vulnerabilidades de grande impacto. As vulnerabilidades de grande impacto são aquelas cujo seu CVSS é igual ou superior a 7.0 e que são detetadas pela totalidade das ferramentas utilizadas durante a análise. Por fim, é dada uma breve explicação sobre qual o procedimento que se deve tomar consoante o valor que foi atribuído ao APK. A proposta de procedimento apresentada é a sugestão de como se deve abordar uma aplicação tendo em conta o seu respetivo nível de segurança. Esta sugestão foi concluída durante a realização desta dissertação. A secção 4.4 apresenta de forma mais detalhada e justificada esta abordagem.

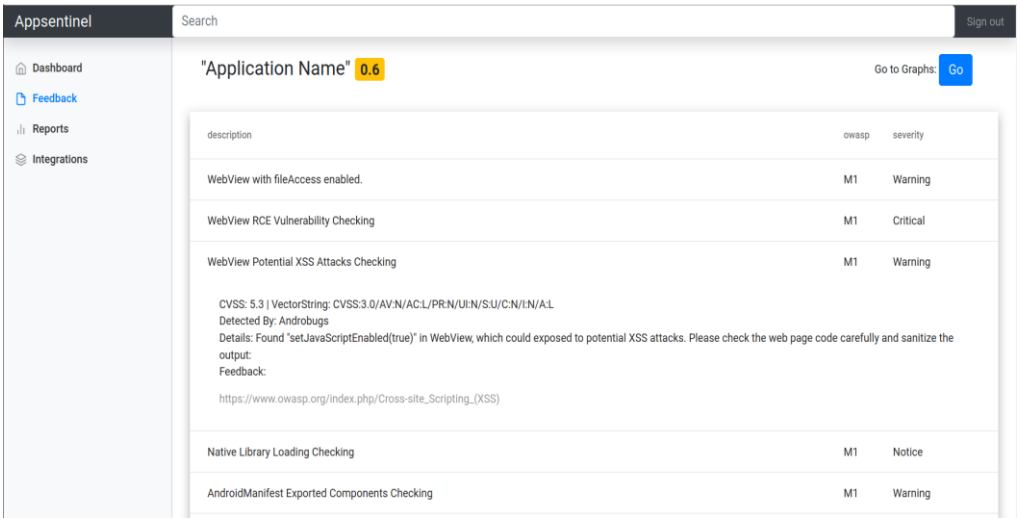

Figura 24 - Imagem da apresentação do relatório de análise do Appsentinel

Outro ecrã de apresentação, é o que mostra o relatório da análise efetuada. Este é constituído por uma tabela que agrega todas as vulnerabilidades detetadas pelo AppSentinel. Cada linha da tabela representa uma vulnerabilidade e apresenta o nome da vulnerabilidade, a respetiva categoria *OWASP* e o respetivo nível de severidade. Também pode ser visualizado cada vulnerabilidade em mais detalhe ao clicar por cima da linha da tabela correspondente. Ao clicar é feita uma expansão da linha contendo mais informações sobre a vulnerabilidade, como o respetivo CVSS quando aplicável, a identificação das ferramentas que identificaram a vulnerabilidade, uma descrição mais detalhada sobre a vulnerabilidade e também é apresentado o feedback onde pode estar a proposta de mitigação. Este caso pode ser visualizado na Figura *25*.

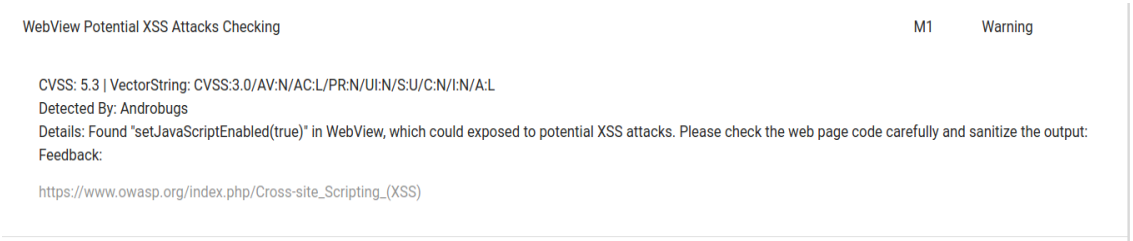

Figura 25 - Informação detalhada da vulnerabilidade

4 Análise e discussão de resultados

Neste capítulo será apresentado um conjunto de testes que foram realizados para este estudo. O objetivo é estudar o comportamento do mecanismo de cálculo para aplicações reais, e analisar e validar os valores de risco calculados. Uma vez que o sistema está preparado para analisar aplicações Android, foram descarregadas quatrocentas e quinze aplicações da loja Aptoide. As aplicações descarregadas correspondem às dez com maior número de downloads por cada categoria existente na loja.

4.1 Conjunto de dados utilizados

Os testes foram executados num único servidor equipado com um processador *AMD Ryzen 7 Pro 3700* processando a 4.4GHz com 64GB de *RAM DDR4 ECC 2666MHz*. Também estava equipado com 2x SSD NVMe 960GB e corria um servidor com o sistema operativo *Ubunbtu 18.04* com o núcleo genérico 4.15.0-109.

Em primeiro lugar, foram analisadas aplicações propositadamente vulneráveis, as DVAA (*Damn Vulnerable Apps Android*). Estas aplicações foram utilizadas com o propósito de validar o sistema uma vez que assim torna-se possível avaliar o comportamento consoante ao número e tipo de vulnerabilidades que devem ser detetadas nestas aplicações.

APKs analisados	Número total de Vulnerabilidades	Número Total Notices	Número Total Warnings	Número Total Criticals	Média de vulnerabilidades por APK	Media do risco
9	145	42	66	37	16	0,51

Tabela 2 - Resultados gerados pela análise aos DVAA

Posteriormente, foi feita uma análise exaustiva a aplicações reais. O sistema foi configurado para usar três ferramentas de análise diferentes, Androbugs, DroidStatX e Super Android Analyzer e a analise feita aos quatrocentos e quinze APK's (descarregados da loja Aptoide) demorou cinquenta e cinco minutos e trinta e nove segundos. Desse número de APK´s analisados, trezentos e oitenta e quatro geraram resultados. Por vezes as ferramentas não conseguem analisar os APK´s isto deve-se principalmente à ofuscação do código.

APKs analisados	Número total de Vulnerabilidades	Número Total Notices	Número Total Warnings	Número Total Criticals	Média de vulnerabilidades por APK	Media do risco
384	11308	3799	4821	2688	29	0,58

Tabela 3 – Resultados gerados pela análise dos 384 APKs

4.2 Testes com DVAA

De forma a validar a solução desenvolvida, foram analisadas aplicações cujo desenvolvimento delas tem o propósito de apresentar vulnerabilidades, conhecidas como *Damn Vulnerable Applications Android* ou DVAA. Deste modo, torna-se possível prever o comportamento do sistema relativamente aos resultados gerados por analisar este tipo de aplicações. No entanto, é preciso ter em conta que este tipo de aplicações são desenvolvidos com propósitos didáticos, ou seja, diferem muito relativamente à implementação de aplicações reais. As DVAAs usadas para estes testes foram as seguintes:

- PIVAA
- Vulnerable Android APK
- DVHMA
- InsecureBankV2
- SIEVE
- OWASP-MSTG UnCrackable Apps

Name	Score	Vulnerabilities	Notice	Warning	Critical
pivaa	0,75	29	6	12	11
Vulnerable APK	0,71	4	0	3	1
InsecureBankv2	0,61	29	9	13	7
de.zertapps.dvhma.openui5_1.0.0_6.3.0_debug	0,58	16	4	8	4
sieve	0,57	19	7	7	5
de.zertapps.dvhma.featherweight_1.0.0_3.5.0_debug	0,54	17	5	7	5
UnCrackable-Level1	0,48	12	4	6	2
UnCrackable-Level2	0,44	9	3	5	1
UnCrackable-Level3	0,42	10	4	5	1

Tabela 4 – Resultados da Análise às DVAAs selecionadas para este caso de estudo.

De acordo com o que pode ser visualizado na Tabela 7, existem apenas duas aplicações que se encontram no grupo de risco elevado, PIVAA e Vulnerable APK. De acordo com a documentação, o PIVAA apresenta cerca de vinte e sete vulnerabilidades de vários níveis de severidade e no teste realizado, apresenta vinte e nove. Ora neste caso é possível concluir que a ferramenta de cálculo atribui um valor de risco aceitável. No caso do Vulnerable APK, embora sejam apresentadas apenas 4 vulnerabilidades após a análise, a vulnerabilidade critica encontrada é detetada pelas três ferramentas em utilização e o seu valor CVSS é muito alto, daí o resultado de valor de risco ser o que é apresentado.

No entanto, para este caso o valor de risco da aplicação poderá não ser tão coerente quando em comparação com o número de vulnerabilidades detetadas em outras aplicações.

Por outro lado, o InsecureBankV2, também é uma das DVAA que apresenta mais vulnerabilidades de acordo com a sua documentação, seria de esperar um valor de risco mais elevado. Isto pode acontecer por falha de deteção de algumas ferramentas, porém o valor de risco atribuído ao InsecureBankV2 pertence ao grupo de risco moderado, querendo isto dizer que a aplicação não é segura e é aconselhável fazer uma revisão ao código.

Por fim, as restantes aplicações apresentam valores de risco de acordo com as expectativas.

4.3 Levantamento de resultados

A análise aos resultados dos testes realizados para este caso de estudo, mostram que existe uma grande quantidade de vulnerabilidades encontradas, onde a média por cada APK é de 29 vulnerabilidades, um número significativo, porém é preciso ter em conta os possíveis falsos positivos. O tipo de vulnerabilidades mais frequentes são as do tipo *Warning* e apresentam um nível de risco médio, porém apelam aos programadores atenção e revisão ao código, neste tipo de vulnerabilidades. Em segundo lugar, as vulnerabilidades mais frequentes são as de risco mais baixo (*Notices*), sendo que as vulnerabilidades de risco mais elevado (*Criticals*) representam 23,77% do total.

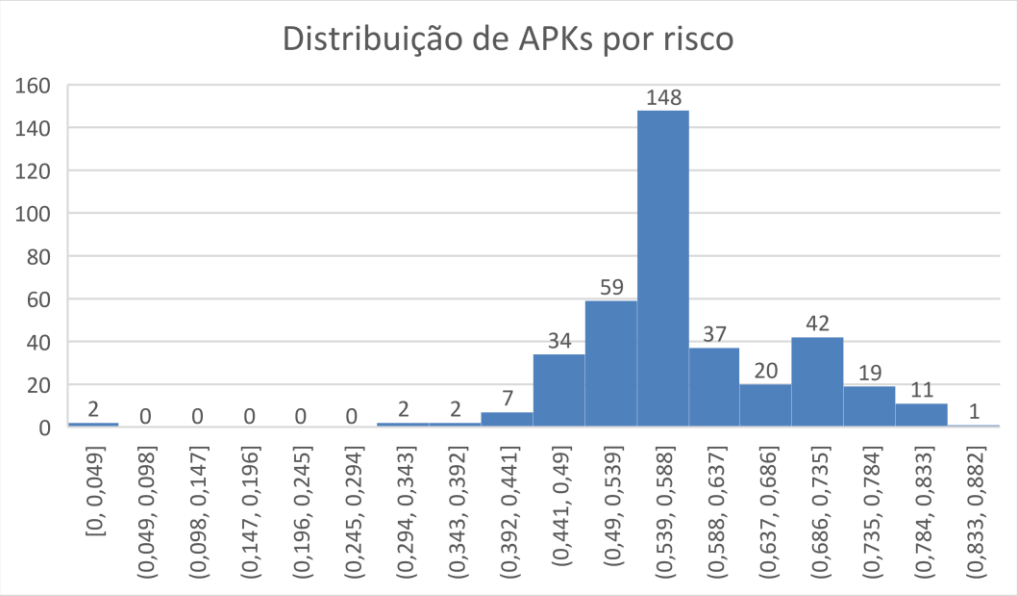

Figura 26 – Gráfico com a distribuição de APKs consoante os seus níveis de risco

De acordo com o apresentado na Figura *26*,vemos que a maioria das aplicações se encontram com valores de risco na casa dos *0,5* sendo valores muito próximos da média. Aplicações que

apresentam estes valores de risco normalmente têm um número significativo de *notices* e de *warnings*, sendo que o número de críticos normalmente não é muito elevado. Porém como existe sempre um peso associado a cada vulnerabilidade, ocorrem casos onde o número de vulnerabilidades é reduzido, mas aplicação pode apresentar um valor de risco idêntico.

É notório que as aplicações tendem mais para valores de risco elevado, sendo que abaixo de 0,4 existem muito poucas aplicações. Á partida, isto quer dizer que a maioria das aplicações apresentam um risco relevante e deve ser tido em conta porque é altamente provável que existam vulnerabilidades que merecem atenção e que obriguem os programadores a rever o seu código.

4.3.1 TOP 15 APKs de menor risco

No seguimento da análise aos resultados gerados neste caso de estudo, foram criadas duas tabelas com as quinze aplicações de maior risco e as quinze de menor risco. Deste modo, torna-se possível ver que tipo de características estas aplicações apresentam e assim entender melhor o porque dos valores calculados.

NAME	BOTTOM 15	Size(bytes)	notices	warnings	criticals
OfficeSuite Pro + PDF	0	39920396	0	0	0
Gojek	0	86431814	0	0	0
Flashlight Widget	0,3	32123	3	3	0
Flashlight Widget - Original	0,33	56594	2	1	0
mobile.de – Germany‘s largest car market	0,35	18649708	7	9	3
Google Currents	0,38	860160	4	1	1
Color Days Widget	0,41	1558802	2	4	0
Moto	0,41	10571369	8	9	1
DigiD	0,41	38255900	9	9	0
Running Pedometer Step Counter	0,43	2367047	7	6	2
idealo - Price Comparison & Mobile Shopping App	0,43	15385745	6	5	1
Classic Text To Speech Engine	0,44	969165	7	5	3
MX Player Codec (ARMv6 VFP)	0,44	5920707	1	1	0
Loopsie - Pixeloop Video Effect & Living Photos	0,45	28982166	5	8	1
DamonPS2 Free - Fastest PS2 Video Games Emulator	0,45	26775896	3	4	1

Tabela 5 – 15 aplicações ordenadas por valor de risco mais baixo. As mais seguras neste caso de estudo

Relativamente às aplicações mais seguras, apresentadas na tabela 4, podemos ver algumas características que podem justificar o facto de elas apresentarem valores de risco baixos. Como por exemplo, o tamanho das aplicações poderá ser um fator influenciável, aplicações menos complexas poderão ter menos vulnerabilidades. Muitas vulnerabilidades estão associadas a características de implementação mais complexas ou da dependência de mais recursos dentro e fora do dispositivo.

Sendo que uma aplicação simples e que use poucos recursos quer internos quer externos, é normal que não apresente grandes riscos.

Contudo, não é só o tamanho do APK que vai definir se uma aplicação é de alto risco ou não. É preciso olhar para o número e tipo de vulnerabilidades encontradas. É possível ver em alguns casos que o número de *warnings* e *notices* não é reduzido, porém é preciso relembrar que cada vulnerabilidade tem um peso associado, para estes casos as vulnerabilidades devem ter pesos relativamente baixos dando origem assim, a um valor de risco baixo. O mesmo aplica-se a aplicações com poucas vulnerabilidades detetadas, mas com valores de risco mais altos, ou seja, o peso das vulnerabilidades é alto. Outra característica relevante que seria de esperar para as quinze aplicações da Tabela 4, é o número reduzido de vulnerabilidades do tipo crítico.

4.3.2 TOP 15 APKS de maior risco

Contrariamente ao analisado na secção anterior, desta vez foram analisadas as quinze aplicações que apresentam maior risco, é importante ver o contraste entre as aplicações de maior risco e de menor risco para tentar perceber o porque do sistema ter atribuído esses valores.

Name	TOP 15	Size(bytes)	notices	warnings	criticals
AnimeDroid	0,87	22146504	14	15	22
ES File Explorer/Manager PRO	0,83	17755157	14	18	19
KineMaster Pro – Editor de Vídeos	0,81	23071212	14	15	17
TouchPal	0,81	46818737	14	18	17
Draw Cartoons	0,81	24882389	15	14	18
City Coach Bus Simulator 17	0,81	55523152	13	17	17
Weather	0,81	17499693	12	22	17
TeaTV	0,8	21315444	13	22	15
KineMaster – Pro Video Editor	0,8	33313010	14	16	16
GO Weather - Widget, Theme, Wallpaper, Efficient	0,8	23406395	15	16	15
Beauty Makeup Selfie Camera MakeOver Photo Editor	0,8	40071813	13	22	8
Facebook Video Downloader	0,79	3231471	10	13	14
Aldiko Book Reader	0,78	17044135	11	15	14
Filters for changing cat face & dog face	0,78	41857412	11	14	14
SURE - Smart Home and TV Universal Remote	0,77	66177278	15	18	15

Tabela 6 - 15 aplicações ordenadas por valor de risco mais alto. As menos seguras neste caso de estudo

Em continuação, desta vez olhando para as aplicações menos seguras vemos que o número de vulnerabilidades encontradas é muito superior comparativamente com as vulnerabilidades encontradas nas aplicações da Tabela anterior. E como seria espectável, o número de vulnerabilidades críticas é bastante relevante sendo que o "*AnimeDroid*" apresenta o maior número de *criticals* em

comparação com todas as outras aplicações deste teste. No entanto, o facto de uma vulnerabilidade ser detetada por mais que uma ferramenta, automaticamente vai incutir um peso maior no cálculo do risco final. Como é o caso de "*Beauty Makeup Selfie Camera MakeOver Photo Editor*", onde apresenta menos vulnerabilidades críticas em comparação com os outros, porém com um valor de risco alto.

Contrariamente ao que foi escrito para a tabela anterior, na Tabela 6 as aplicações apresentam um tamanho maior querendo isto dizer que o seu nível de complexidade e dependências de recursos também é maior. Quando se faz uso de mais recursos e aumenta-se a complexidade de uma aplicação, também é preciso aumentar o nível de segurança da mesma. Caso isso não aconteça, os dados dos utilizadores correm um risco maior de serem comprometidos.

Em seguida, também foi feita uma análise aos 20 APKs mais descarregados da loja Aptoide para ver o nível de risco das aplicações mais conhecidas. É importante esta análise pois são as aplicações que mais contém informações de utilizadores.

4.3.3 TOP 20 APKs mais descarregados

Por último, foi decidido analisar os APKs mais descarregados. Estes são as aplicações mais populares da loja Aptoide e são os que aparentemente mais residem nos dispositivos dos utilizadores. Por isso foi feito esta categorização para deduzir o risco que os utilizadores podem estar a correr ao utilizar estas aplicações.

Name	TOP 20 (downloads)	score
Google Camera	376096683	0,5
Skins Editor for Minecraft PE (3D)	216031302	0,55
WhatsApp Messenger	210622718	0,57
Facebook	146770992	0,55
Duo Mobile	134934621	0,49
Polarr Photo Editor	124624854	0,57
Messenger – Text and Video Chat for Free	106048407	0,57
Torch Flashlight LED HD	98388644	0,6
Instagram	80304625	0,68
Flashlight Widget	78392590	0,3
Carrier Services	74706216	0,57
FlightStats	59778124	0,58
BeyondPod Podcast Manager	48799190	0,56
RelaxBanking Mobile	46630816	0,72
Snapchat	45793559	0,53
MEB E-OKUL VBS	45277313	0,53
VIP Access	44362402	0,59
Share Link – File Transfer	43544158	0,69

idealo - Price Comparison & Mobile Shopping App	41124655	0,43
QRbot: QR code reader and barcode reader	40138003	0,54
	média:	**0,56**

Tabela 7 – As 20 aplicações mais descarregadas da loja Aptoide e respetivos valores de risco

De acordo com os dados apresentados na Tabela *7* – As 20 aplicações mais descarregadas da loja Aptoide e respetivos valores de risco, podemos ver que maior parte das aplicações apresentam valores de risco moderados com uma média de 0,56. Seria de esperar que aplicações como, WhatsApp, Facebook, Instagram, Snapchat apresentassem valores de risco baixos. São as aplicações mais utilizadas por todo o mundo e devem estar bem implementadas e de forma segura. Há exceção do Instagram, as restantes aplicações mencionadas anteriormente apresentam um risco moderado com valores na casa das cinco décimas. Uma vez que se trata de aplicações complexas e com um uso relativo de permissões é normal que apresentem vulnerabilidades. No caso do Instagram, o valor já é mais alto, deveria ser tomada uma revisão no código, porém não alarmante.

4.4 Conclusões

Sabemos da existência de grupos de vulnerabilidades separados por *notices*, *warnings* e *criticals*, faria sentido inserir uma aplicação num determinado grupo consoante o seu nível de risco. Assim é possível saber em que grupo uma aplicação se encontra e quais características apresenta relativamente à quantidade e tipo de vulnerabilidades. Este tipo de agrupamento pode ser essencial na gestão de risco, pode ser a chave da decisão de publicar uma aplicação ou não. O agrupamento de aplicações consoante o nível de risco foi feito da seguinte forma:

- Grupo Risco Baixo: Valores de risco no intervalo pertencente a [0;0,5[. Estes tipos de aplicações não apresentam risco para o utilizador final, porém mitigar as vulnerabilidades encontradas é sempre uma boa prática. As vulnerabilidades mais frequentes são do tipo *warning* e *notice*, as do tipo *critical* são menos comuns.
- Grupo Risco Moderado: Valores de risco pertencentes ao intervalo [0,5;0,7[. Aplicações deste grupo não apresentam grande risco para o utilizador final, porém devem ser revistas, principalmente para mitigar vulnerabilidades do tipo critico. Estas aplicações normalmente apresentam um número significativo de *warnings* e *notices*, também apresentam *criticals*, porém não tão significativos.
- Grupo Risco Elevado: Valores de risco pertencentes ao intervalo [0,7;1]. Estas aplicações podem apresentar um sério risco ao utilizador final e devem ser revistas antes de serem

publicadas na loja. Apresentam números significativos para a quantidade de vulnerabilidades do tipo *notice, warning e critical*.

Em seguida, foram agrupados os 384 APKs para os grupos respetivos aos seus valores de risco. A taxa de risco para este caso de estudo pode ser visualizada na Figura *27*.

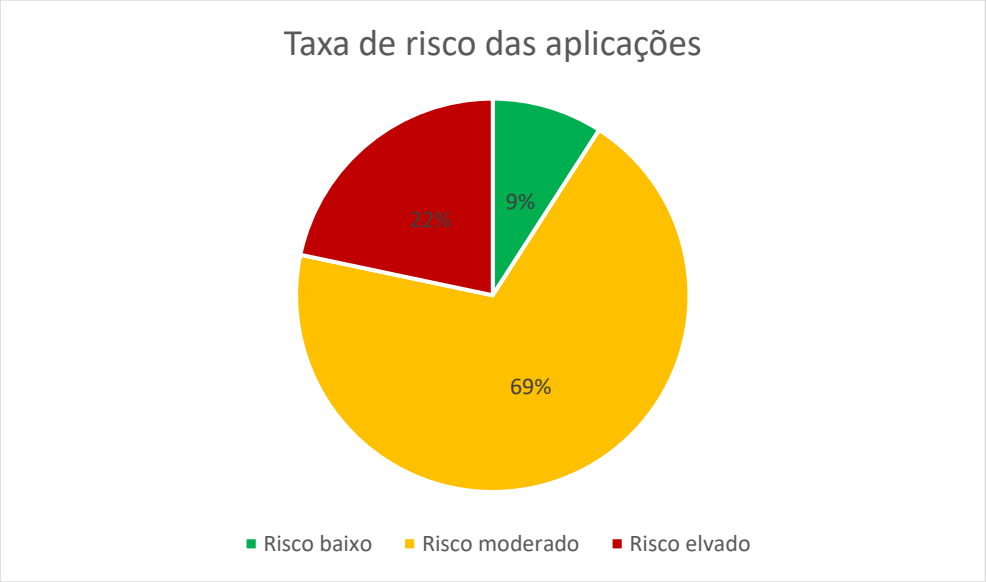

Figura 27 – Gráfico com a taxa de risco das aplicações.

De acordo com o que pode ser visualizado na Figura *27* e de acordo com o sistema de cálculo desenvolvido nesta dissertação, podemos concluir que grande parte do desenvolvimento de aplicações android ainda não está focado na segurança. A segurança no sistema Android tem vindo a melhorar ao longos dos anos, a atualização do sistema para versões superiores é responsável por isso. Mesmo assim, as aplicações desenvolvidas para este sistema operativo, não apresentam os melhores resultados a nível de segurança.

No entanto, a análise foi feita com base nos testes realizados para estas aplicações. Estes testes dependem do sistema desenvolvido e este depende de ferramentas externas desenvolvidas por outros autores. Estas ferramentas podem apresentar falsos positivos e falsos negativos durante as suas análises. O risco real de uma aplicação pode ser diferente do risco calculado por este sistema, mesmo assim, os resultados mostram ser bastante apelativos e devem ajudar uma equipa de desenvolvimento a identificar melhor problemas relacionados com a segurança de uma aplicação Android assim como *QA managers* de lojas de aplicações na tomada de decisão para aceitar ou rejeitar aplicações na loja.

5 Conclusão e Trabalhos Futuros

Neste capítulo serão mencionadas as principais conclusões desta dissertação e também serão indicadas sugestões de trabalho futuro de modo a permitir que este estudo seja continuado. Com a esperança que os trabalhos futuros possam contribuir para a evolução deste sistema e que o cálculo de risco de uma aplicação seja mais preciso e que defenda a sua importância durante o desenvolvimento.

Ao longo desta dissertação ainda foram escritos dois artigos científicos, um dos quais que chegou a ser publicado, "*Automated security testing of Android applications for secure mobile development*", e que foi apresentado na *2020 IEEE International Conference on Software Testing, Verification and Validation (ICST 2020) ICST.* O outro artigo já foi submetido e está a ser avaliado.

5.1 Conclusão

Esta dissertação tinha como principal objetivo desenvolver um processo de cálculo automático para avaliar o risco de aplicações Android utilizando mecanismos de análise estática. E, desta forma, ser possível contribuir para um desenvolvimento seguro e para a educação dos programadores relativamente aos riscos em ambiente Android. Por isso, foi proposto e desenvolvido um sistema, com recurso à metodologia *Design Science Research*, para dar resposta à questão de investigação. A revisão da literatura foi uma fase importante pois permitiu aglomerar um vasto conjunto de conhecimento essencial para cumprir o objetivo definido. O método de cálculo do CVSS e as ferramentas de análise de vulnerabilidades tiveram um papel importante para o desenvolvimento desta solução. Foi difícil encontrar projetos e artigos associados a este problema mesmo assim, trabalhos anteriores mostraram que adotar o CVSS para quantificar riscos associados a vulnerabilidades é a abordagem mais correta.

No entanto, o sistema de cálculo do AppSentinel cumpre o objetivo definido e contribui para ajudar os programadores e lojas de aplicações a fazer uma melhor gestão de risco. Um programador ao submeter a aplicação desenvolvida neste sistema de análise, automaticamente saberá se deve publicar uma aplicação ou não com base no risco calculado. O mesmo acontece para uma loja de aplicações, que ao definirem intervalos limitativos, podem rejeitar ou aceitar aplicações. As lojas também podem notificar as entidades responsáveis pelo desenvolvimento para melhorarem a questão da segurança no código implementado em suas aplicações, e ao longo do tempo podem ir controlando o efeito resultado desse pedido através do sistema de cálculo de risco. Desta forma é possível aplicar a solução para várias situações práticas e ainda mais poder contribuir para a segurança do mundo virtual que está em alto crescimento ano após ano.

Relativamente aos resultados obtidos nos testes efetuados, pode concluir-se que em geral as aplicações não estão desenvolvidas de forma muito segura. A verdade é que a maioria das aplicações não apresenta um elevado risco para o utilizador, mas o número de aplicações efetivamente seguras é muito reduzido. Isto mostra que a segurança não é um fator primordial durante o desenvolvimento de uma aplicação Android. A segurança *mobile* ainda não é tão aprofundada como a segurança nas aplicações *web*, porém é necessário apostar mais num desenvolvimento seguro para aplicações móveis porque os *smartphones* armazenam e manipulam muita informação sensível dos utilizadores.

Este trabalho foi desenvolvido de forma integrada no projeto AppSentinel, cofinanciado pelo Lisboa2020/Portugal2020/EU no contexto do Sistema de Incentivos à I&DT - Projetos em co promoção (project 33953).O código do projeto encontra-se disponível no seguinte repositório online: https://github.com/pontocom/appsentinel.

5.2 Trabalhos Futuros

A segurança da informação será sempre um tópico de constante evolução. Ao longo do tempo surgem sempre novas tecnologias com características nunca estudadas do ponto de vista da segurança. Isto exige que os profissionais de tecnologias de informação estejam sempre atualizados a nível de conhecimento. Outra questão importante é a evolução do conhecimento da população em geral relativamente aos sistemas de informação, há cada vez mais pessoas com conhecimentos nesta área e uma parte delas com intenções maliciosas. Desta forma surgirão mais ataques no mundo digital o que requer por parte das organizações um investimento na área da segurança. É nesta matéria que entra esta dissertação como meio de prevenção para potenciais riscos ou prejuízos.

Contudo, a ferramenta de cálculo de risco automático é uma ideia que claramente contribui para um ciclo de desenvolvimento seguro, assim como o sistema AppSentinel. Porém, devido a situação que se viveu este ano e tendo em conta os limites temporais associados a este tipo de trabalhos, nem todas as ideias foram adotadas de forma a concretizar um sistema o mais perto possível da perfeição. Também foram identificadas algumas limitações da solução, o sistema nem sempre deteta todas as vulnerabilidades e pode apresentar um número de falsos positivos que influencia o valor de risco calculado sendo este não coerente com o nível de segurança real de uma aplicação. Por isso, um trabalho futuro poderia ser focado no aperfeiçoamento desta ferramenta.

Neste momento, o sistema está desenvolvido de forma a calcular o risco dependendo de um conjunto de ferramentas externas, estas ferramentas demoram um certo tempo ao processar a sua análise. Este tempo de processamento pode ser melhorado se o sistema for distribuído, por outras palavras, cada ferramenta podia correr num servidor independente e deste modo o tempo de análise seria drasticamente reduzido. Tendo em conta que o sistema está preparado para adicionar mais

plugins e sendo esse o objetivo deste sistema, esta melhoria seria um alto contributo para aumentar a sua robustez.

Outra limitação da solução apresentada nesta dissertação é o facto do cálculo de risco não ser separado por versões Android, querendo isto dizer que o valor de risco de uma aplicação pode variar consoante a versão do sistema operativo. Existem vulnerabilidades que, para certas versões Android, já não podem ser exploradas e assim certos valores atribuídos a essas vulnerabilidades podem estar desatualizados.

6 Bibliografia

Alenezi, M., & Almomani, I. (2018). 5th International Symposium on Data Mining Applications, SDMA 2018. In *Advances in Intelligent Systems and Computing* (Vol. 753, Issue March). Springer International Publishing. https://doi.org/10.1007/978-3-319-78753-4

Altuwaijri, H., & Ghouzali, S. (2020). Android data storage security: A review. *Journal of King Saud University - Computer and Information Sciences*, *32*(5), 543–552. https://doi.org/https://doi.org/10.1016/j.jksuci.2018.07.004

Chin, E., Felt, A. P., Greenwood, K., & Wagner, D. (2011). Analyzing inter-application communication in Android. *MobiSys'11 - Compilation Proceedings of the 9th International Conference on Mobile Systems, Applications and Services and Co-Located Workshops*, 239–252. https://doi.org/10.1145/1999995.2000018

Corral, L., & Fronza, I. (2015). Better Code for Better Apps: A Study on Source Code Quality and Market Success of Android Applications. *Proceedings - 2nd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2015*, 22–32. https://doi.org/10.1109/MobileSoft.2015.10

CVE. (2018). *History*. https://cve.mitre.org/about/history.html

F5. (2016). *F5 BIG-IP Platform Security*. https://www.f5.com/services/resources/white-papers/f5-big-ip-platform-security

Fang, Z., Han, W., & Li, Y. (2014). Permission based Android security: Issues and countermeasures. *Computers and Security*, *43*(0), 205–218. https://doi.org/10.1016/j.cose.2014.02.007

Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. *Proceedings of the ACM Conference on Computer and Communications Security*, 627–636. https://doi.org/10.1145/2046707.2046779

FIRST. (2019). *Common Vulnerability Scoring System version 3.1 Specification Document Revision 1*. 1–24. https://www.first.org/cvss/

Gandhewar, N., & Sheikh, R. (2010). Google Android : An Emerging Software Platform For Mobile Devices. *International Journal on Computer Science and Engineering*, *1*(6), 12–17. https://doi.org/10.1007/s003810050241

Google. (n.d.). *Manifest.permission_group*. Retrieved January 2, 2020, from https://developer.android.com/reference/android/Manifest.permission_group

Khazaei, A., Ghasemzadeh, M., & Derhami, V. (2016). An automatic method for CVSS score prediction using vulnerabilities description. *Journal of Intelligent and Fuzzy Systems*, *30*(1), 89–96. https://doi.org/10.3233/IFS-151733

Kulkarni, K., & Javaid, A. Y. (2018). Open Source Android Vulnerability Detection Tools: A Survey. *ArXiv Preprint ArXiv:1807.11840*.

Li, L., Gao, J., Hurier, M., Kong, P., Bissyandé, T. F., Bartel, A., Klein, J., & Traon, Y. le. (2017). *AndroZoo++: Collecting Millions of Android Apps and Their Metadata for the Research Community*. 276–277. http://arxiv.org/abs/1709.05281

Merkel, D. (2014). Docker : Lightweight Linux Containers for Consistent Development and Deployment Docker : a Little Background Under the Hood. *Linux Journal*, *2014*(239), 2–7. http://delivery.acm.org.ezproxy.library.wisc.edu/10.1145/2610000/2600241/11600.html?ip=128.104.46.196&id=2600241&acc=ACTIVE SERVICE&key=066E7B0AFE2DCD37.066E7B0AFE2DCD37.4D4702B0C3E38B35.4D4702B0C3E38B35&__acm__=1557803890_216b4a0168a6b29b8f2e7a74

Microsoft. (2018). *Threat modeling for drivers*. Threat modeling for drivers

Nancy R. Mead, Julia H. Allen, Sean Barnum, Robert J. Ellison, G. R. M. (2008). Software Security Engineering: A Guide for Project Managers. In *Software Security Engineering: A Guide for Project Managers* (pp. 84–92). Addison-Wesley. https://books.google.pt/books?id=hl-zvFPQAfcC&printsec=frontcover&dq=Software+Security+Engineering:+A+Guide+for+Project+Managers&hl=pt-PT&sa=X&ved=0ahUKEwj0pcGLjf7mAhVMOhoKHRCQDCUQ6AEIKTAA#v=snippet&q=risk assessment&f=false

NIST. (2019). *NVD Data Feeds*. https://nvd.nist.gov/vuln/data-feeds

OWASP. (2019). *No Title*. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Petraityte, M., Dehghantanha, A., & Epiphaniou, G. (2018). A model for android and iOS applications risk calculation: CVSS analysis and enhancement using case-control studies. In *Advances in Information Security* (Vol. 70, pp. 219–237). Springer New York LLC. https://doi.org/10.1007/978-3-319-73951-9_11

Rahman, A., Pradhan, P., Partho, A., & Williams, L. (2017). Predicting Android Application Security and Privacy Risk with Static Code Metrics. *Proceedings - 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2017*, 149–153. https://doi.org/10.1109/MOBILESoft.2017.14

Rao, K. R. M., Division, G., & Pant, D. (2010). *Security risk assessment of Geospatial Weather Information System (GWIS) : An OWASP based approach*. *8*(5).

Robinson, G., & Weir, G. R. S. (2015). Understanding android security. *Communications in Computer and Information Science*, *534*, 189–199. https://doi.org/10.1007/978-3-319-23276-8_17

Scarfone, K., & Mell, P. (2009). An analysis of CVSS version 2 vulnerability scoring. *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, 516–525. https://doi.org/10.1109/ESEM.2009.5314220

Statcounter. (2019). *Mobile Operating System Market Share Worldwide*. https://gs.statcounter.com/os-market-share/mobile/worldwide

Walsh, S. (1983). Software security. *Data Processing*, *25*(3), 9–10. https://doi.org/10.1016/0011-684X(83)90093-X

Wang, J. A., Wang, H., Guo, M., & Xia, M. (2009). Security metrics for software systems. *Proceedings of the 47th Annual Southeast Regional Conference, ACM-SE 47*. https://doi.org/10.1145/1566445.1566509

Wang, Y., Zheng, J., Sun, C., & Mukkamala, S. (2013). Quantitative security risk assessment of Android permissions and applications. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7964 LNCS*(September 2012), 226–241. https://doi.org/10.1007/978-3-642-39256-6_15

Zhou, Y., & Jiang, X. (2012). Dissecting Android malware: Characterization and evolution. *Proceedings - IEEE Symposium on Security and Privacy*, *4*, 95–109. https://doi.org/10.1109/SP.2012.16

2.1: Activities and intents. (2020). Consultado em Setembro 28, 2020, de https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-1-get-started/lesson-2-activities-and-intents/2-1-c-activities-and-intents/2-1-c-activities-and-intents.html

Configure your build ： Android Developers. (2020, August 26). Consultado em Setembro 28, 2020, de https://developer.android.com/studio/build

Common Vulnerability Scoring System Version 3.1 Calculator. (2018). Consultado em Setembro 28, 2020, de https://www.first.org/cvss/calculator/3.1

Wikipedia (2020, May). Consultado em Setembro 28, 2020 de https://pt.wikipedia.org/wiki/APK

Github, (2018). Consultado em Setembro 28, 2020 de https://github.com/SUPERAndroidAnalyzer/super

Flask. (2020). Consultado em Setembro 28, 2020, de https://palletsprojects.com/p/flask/