INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Análise, deteção e classificação de vulnerabilidades em aplicações Android usando o OWASP Mobile Top 10

Francisco Livramento da Palma

Mestrado em Engenharia de Telecomunicações e Informática

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

Novembro, 2020

Agradecimento

Gostava de agradecer ao meu orientador e professor Carlos Serrão, por me ter convidado para este projeto, por todo o tempo e empenho a que se dedicou para me ajudar em momentos difíceis no desenvolvimento do mesmo e pelo à vontade que me deixou para poder resolver qualquer questão relacionada, mas acima de tudo deixar uma palavra de apreço pela pessoa que é, sempre disposto a ajudar.

Aos meus colegas quero agradecer pela motivação extra me passaram para ultrapassar certas dificuldades.

Quero agradecer à minha família e amigos pela sua compreensão e que sempre me deram força para continuar e evoluir.

Por fim um obrigado a todas as pessoas que fizeram parte da minha vida durante o desenvolvimento da minha dissertação!

Resumo

O desenvolvimento de aplicações móveis em segurança está cada vez mais a ter um papel importante no mundo das tecnologias. As plataformas de distribuição de aplicações para dispositivos móveis têm todos os dias um grande número de aplicações a serem inseridas na sua loja, e muitas delas não são examinadas quanto à existência de falhas de segurança. Estas falhas podem acontecer devido à falta de conhecimentos teóricos ou até mesmo a falta de boas práticas de segurança no desenvolvimento das aplicações móveis. Por consequência, as aplicações que são disponibilizadas aos utilizadores finais poderão estar comprometidas, ou seja, a existência de vulnerabilidades é bastante acentuada, fazendo com que possíveis atacantes intercetem informações confidenciais do utilizador.

Com o objetivo de reduzir o número de aplicações com falhas de segurança e consciencializar e educar os programadores surge esta dissertação, que desenvolveu um sistema capaz de encontrar vulnerabilidades em aplicações *Android*, utilizando várias ferramentas de análise estática e que tem a finalidade de apresentar um relatório detalhado com todas as vulnerabilidades identificadas por todas as ferramentas, categorizadas e normalizadas, e por fim para cada vulnerabilidade disposta no relatório existirá um conjunto de informações educacionais (vídeos, livros, artigos ou outros conteúdos) que ajudarão o programador a poder corrigir ou mitigar a sua aplicação *Android*.

Palavras-chave: segurança, ferramentas, vulnerabilidades, plugins, relatório, aplicações Android

Abstract

The development of mobile applications in security is increasingly playing an important role in the world of technologies. Application distribution platforms for mobile devices have a large number of applications to be inserted in your store every day, and many of them are not examined for security breaches. These failures can happen due to the lack of theoretical knowledge or even the lack of good security practices in the development of mobile applications. As a result, the applications that are made available to end users may be compromised, that is, the existence of vulnerabilities is quite pronounced, making it possible for attackers to intercept confidential user information.

In order to reduce the number of applications with security flaws and to raise awareness and educate programmers, this dissertation emerged, which developed a system capable of finding vulnerabilities in Android applications, using various static analysis tools and which has the purpose of presenting a detailed report with all the vulnerabilities identified by all the tools, categorized and normalized, and finally for each vulnerability displayed in the report there will be a set of educational information (videos, books, articles or other content) that will help the programmer to be able to correct or mitigate their Android application.

Keywords: security, tools, vulnerabilities, plugins, report, Android applications

Índice

Índice de Figuras

Índice de Tabelas

Índice de Gráficos

CAPÍTULO 1

Introdução

1.1. Motivação e Enquadramento

O número de utilizadores de dispositivos móveis está a crescer a uma velocidade enorme e a realidade dos tempos modernos coincide com uma diferente abordagem da definição de um telemóvel, em que apenas servia para efetuar chamadas telefónicas e/ou receber mensagens. Nesta nova era, o *smartphone* tem muitas mais funcionalidades do que apenas as referidas em cima. Existem milhares de aplicações que tornam o quotidiano mais fácil, como por exemplo aplicações bancárias (onde se pode fazer transferências, pagamentos e outros), aplicações de saúde e *fitness* (monitorizar, classificar e dar sugestões ao utilizador, são algumas das características destas categorias), aplicações de comércio, e muitas outras categorias estão a ser desenvolvidas neste exato momento, para dar resposta à crescente utilização de *smartphones*.

Os principais sistemas operativos que comandam o mercado dos dispositivos móveis são *Android* e *iOS* (Xanthopoulos & Xinogalos, 2013).

Contudo, como o número de utilizadores cresce e a quantidade de informação sensível nas plataformas é colocada, elas tornam se mais apetecíveis a atacantes, que tentarão ganhar o acesso ao dispositivo ou roubar informação pessoal. E com o fácil acesso à informação, um utilizador malicioso não precisa ter grandes conhecimentos para atacar um dispositivo. Estas plataformas para dispositivos móveis, devido à sua popularização, possuem um elevado número de programadores, contudo surge um grande risco, um largo número de erros de desenvolvimento irá aparecer com maior frequência, isto refletirá no aparecimento de vulnerabilidades que poderão ser exploradas pelos atacantes. Para antecipar o surgimento destas vulnerabilidades, é necessário ter cuidado com o desenvolvimento das aplicações, pois grande parte dos programadores têm bases sólidas e conhecimentos técnicos, mas apresentam dificuldade no entendimento de riscos no desenvolvimento de *software*.

Atualmente, os programadores têm a possibilidade de usar inúmeras ferramentas para analisar e descobrir se as suas aplicações contém vulnerabilidades, que poderão pôr em causa a segurança e informação privilegiada do utilizador final. Contudo, a complexidade na utilização destas ferramentas e os diferentes resultados que as mesmas podem dar, pode colocar o programador numa posição em que não vai dar a importância necessária na descoberta de falhas na sua aplicação.

Considerando estes problemas de segurança (Becher et al., 2011), este projeto vai ajudar os programadores a descobrir, analisar e perceber se as suas aplicações móveis são vulneráveis quando submetidas numa *App Store* (funcionando como um filtro, antes da aplicação estar

acessível) e diminuir o número de ferramentas para a análise de vulnerabilidades tornando o processo mais simples e intuitivo, podendo logo de seguida receber toda a informação detalhada de todas as vulnerabilidades e riscos de uma aplicação. Esta informação poderá ajudar a aumentar a segurança das aplicações móveis antes de a mesma estar disponível para ser transferida para os *smartphones* dos utilizadores finais. E para ter resultados mais concisos e robustos, a utilização de algoritmos de similaridade de texto poderão ser uma opção viável, pois informações com o mesmo conteúdo serão unificadas.

Atualmente existe um grande número de escolhas relativamente às *App Stores*, e são estas que fazem a ligação dos programadores de aplicações móveis e o utilizador final. Também têm um papel importante no que diz respeito à segurança tanto das aplicações que disponibilizam, como na segurança dos *smartphones* dos utilizadores, portanto, devem assegurar confiança e ter responsabilidade pelas suas aplicações. Muitas das *App Stores*, colocam em prática esse papel, combatendo utilizadores mal-intencionados, passando as suas aplicações móveis por vários testes utilizando ferramentas de *antivírus* e *anti-malware*, contudo não há uma grande preocupação no que consiste em dar ao programador boas práticas no desenvolvimento de *software* e de segurança.

Este trabalho foi desenvolvido no contexto de um projeto conjunto entre o Iscte – Instituto Universitário de Lisboa e a empresa Aptoide (que tem a preocupação de assegurar aos seus utilizadores que todas as aplicações são seguras), cujo nome do projeto é AppSentinel (ISCTE-IUL and Aptoide, 2019). A Aptoide é uma *App Store Android*, com mais de 200 milhões de utilizadores em todo o mundo, mais de 6 biliões de *downloads* e cerca de um milhão de aplicações. A abordagem é completamente diferente do habitual, pois cada programador ou empresa pode criar e partilhar a sua própria loja de aplicações. Segundo um estudo a Aptoide foi considerada *a App Store* mais segura entre outras 24 *App Stores* (Ishi et al,.2017).

Dar *feedback* aos programadores de como desenvolver as suas aplicações em segurança e ensinar boas práticas no desenvolvimento de *software* pode ser uma solução para travar muitos riscos que as aplicações possam ter antes de estas estarem disponíveis para o utilizador final.

1.2. Questões de Investigação

Esta dissertação pretende dar apoio a programadores de aplicações *Android*, tendo como foco a segurança das aplicações móveis, dos dispositivos e do próprio utilizador. Os ataques a estes estão a crescer (Palmer, 2019) para tal o combate a este problema passará por analisar e detetar vulnerabilidades mesmo antes de as aplicações estarem disponíveis para serem transferidas para os dispositivos dos utilizadores finais e ajudar o programador a desenvolver aplicações móveis em segurança.

A questão de investigação formulada pretende dar resposta à seguinte questão: será possível desenvolver um sistema automático que ajude os programadores a identificar e corrigir os problemas de segurança das aplicações móveis para aumentar a segurança das mesmas?

1.3. Objetivos

Atualmente a segurança é uma preocupação, tanto dos dispositivos móveis como dos próprios utilizadores e tendo isto em consideração, esta dissertação tem como principal objetivo reduzir o número de aplicações móveis com problemas de segurança mesmo antes de serem colocadas nas *App Stores*. O acesso indesejado de um atacante ou até mesmo a perda de informação delicada, são alguns dos ataques mais comuns nos *smartphones*.

Este trabalho também tem como objetivo criar um sistema capaz de analisar e detetar vulnerabilidades no desenvolvimento de aplicações móveis, através da integração de várias ferramentas de busca de vulnerabilidades. Estudar o comportamento destas ferramentas e perceber se é possível incluí-las ou não neste projeto.

E por fim fornecer um documento com informação detalhada aos programadores. Este documento servirá para ajudar os programadores perceberem os problemas de segurança que as suas aplicações possam estar a enfrentar.

1.4. Método de Investigação

O método de investigação a ser utilizado nesta dissertação é o *Design Science Research*. Esta metodologia "refere-se a um conjunto de orientações e métodos específicos para o processo de criação, construção e validação de um artefacto no contexto de inovação das tecnologias de informação." (Wang e Wang, 2013). As fases deste método de investigação (K. Peffers , T. Tuunanen, 2007) são:

1. Identificação do problema e motivação – Nesta fase serão identificados: o problema e a motivação (referido no capítulo motivação e enquadramento).
2. Objetivos da solução – Será definida para que aplicações este projeto poderá ser aplicado.
3. Desenho e desenvolvimento – Para esta fase será apresentada o desenvolvimento do protótipo.
4. Demonstração – Nesta fase deve ser feito a demonstração do sistema.
5. Avaliação – A aplicação será testada para concluir se serve para o propósito.
6. Comunicação – Se não houver melhorias a fazer, a escrita da dissertação é o próximo passo.

1.5. Estrutura do documento

Este documento está dividido em cinco capítulos de forma a conseguir poder ser explicado todos os conteúdos com rigor. No primeiro capítulo contém uma breve introdução sobre a atualidade dos *smartphones* e das suas aplicações, é explicado quais os objetivos para esta dissertação e qual método de investigação. No segundo capítulo, onde é efetuada a explicação do estado de arte e é explicado mais a fundo alguns dos conceitos mais importantes, como a segurança nos *smartphones*, algumas estatísticas, ferramentas de segurança de aplicações, algumas iniciativas na área da segurança das aplicações móveis, são identificados algoritmos de similaridade de texto e por fim alguns trabalhos relacionados. No terceiro capítulo, é apresentado todo o desenvolvimento do protótipo, descrevendo a sua arquitetura, o funcionamento e descrição de todo o processo. No quarto capítulo descreve os testes realizados e os comentários para cada teste obtido. Por fim, no quinto capítulo, são levantadas as principais conclusões dos resultados desta dissertação levando a apresentar a algum trabalho futuro.

CAPÍTULO 2

Revisão da Literatura

2.1. Introdução

Durante este capítulo será apresentado o estado de arte, onde será abordado o tema da segurança e explicado todos os conceitos, ferramentas e algoritmos utilizados para normalização dos dados.

2.2. Android

Android é um sistema operativo para dispositivos móveis, o seu sistema é baseado em Linux e é um *software open source*. Foi desenvolvido por um consórcio de empresas do ramo das tecnologias cujo nome é Open Handset Alliance e foi lançado oficialmente em 2008. É um software que gere todos os processos do *smartphone*, e também fornece uma interface gráfica para que seja possível a sua utilização.

Atualmente qualquer *smartphone Android* contém uma vasta quantidade de aplicações, jogos, GPS entre outros.

2.2.1 Android SDK

Android SDK é o *software* fornecido para o desenvolvimento de aplicações em *Android* (Cochereau, 2008). Possui um conjunto de ferramentas para serem usadas pelos programadores no desenvolvimento das suas aplicações. Algumas das suas ferramentas são: bibliotecas, *debug* (ferramenta para detetar erros no código), um emulador para correr a aplicação, API e vários tutoriais sobre desenvolvimento de aplicações.

2.2.2 Arquitetura Android

O software *Android* pode ser divido em 5 categorias (Cochereau, 2008), como mostra a Figura 1:

- Aplicações
- *Framework*
- Bibliotecas
- *Android runtime*
- *Linux Kernel*

As aplicações são a camada superior da arquitetura *Android*. Nesta camada é onde estão as aplicações pré-instaladas como os contactos, a câmara, a galeria de fotografia e outras aplicações que possam ser instaladas a partir de uma *App Store*.

Framework fornece conteúdos importantes para criar uma aplicação *Android*. Dispõe de diferentes serviços, por exemplo*, Activity Manager, Content Providers ee Telephony Manager*.

Android runtime é um ambiente que ajuda a camada *Framework* com algumas bibliotecas fundamentais. Essas bibliotecas permitem que as aplicações *Android* possam ser desenvolvidas tanto por *Java* ou *Kotlin*. Contém também uma máquina virtual com o nome Dalvik Virtual Machine, onde é utilizada para correr ao mesmo tempo múltiplas instâncias com eficiência, pois requer pouca memória.

Na camada bibliotecas é onde existem as bibliotecas que fornecem suporte a desenvolvimento *Android*, algumas delas são:

- *Media* - biblioteca que dá suporte a aplicações de música e vídeo.
- *Surface Manager*- biblioteca para gerir o acesso a componentes gráficas.
- *SGL* e *OpenGL* – biblioteca para conteúdos gráficos 2D e 3D.
- *SQLite* – fornece suporte a base de dados
- *SSL* – tecnologia de segurança para estabelecer e encriptar ligações a servidores, por exemplo.

Linux Kernel é a camada onde é gerido todo o hardware, por exemplo:

- *Bluetooth*
- Coluna de áudio
- Memórias

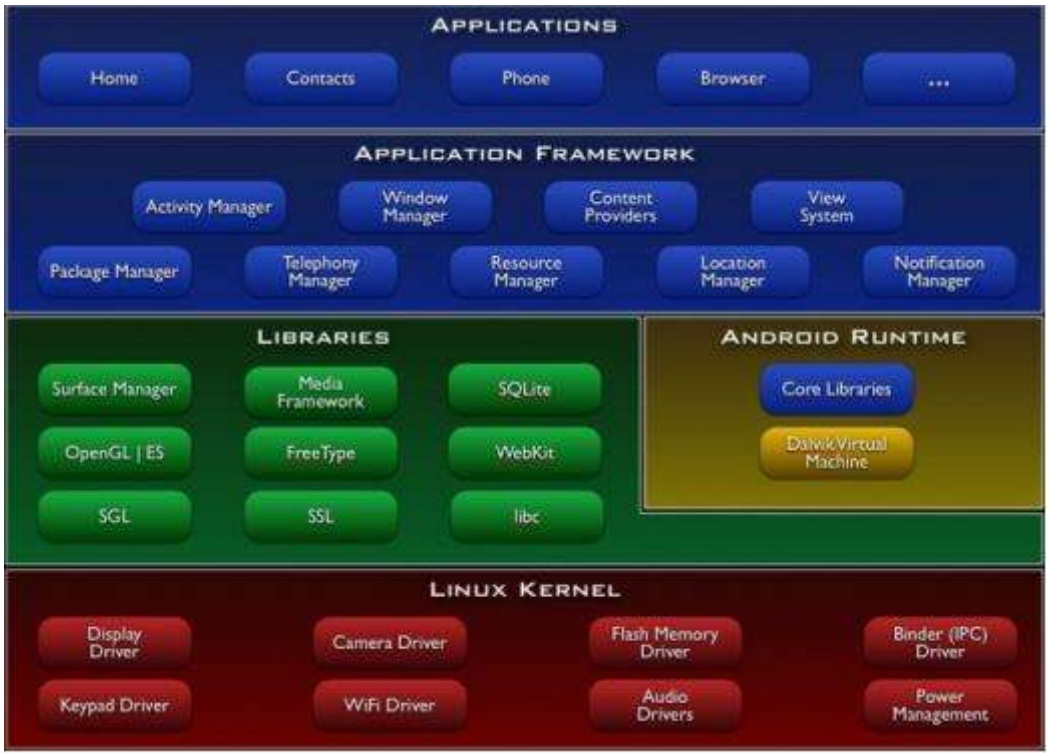

Figura 1-Arquitetura Android (Cochereau,2008)

2.2.3. Principais componentes de uma aplicação Android

A estrutura de uma aplicação *Android* pode dividir se em quatro componentes (Brahler, 2010):

- *Activity*
- *Service*
- *BroadcastReceiver*
- *ContentProvider*

Activity é um ecrã de uma aplicação, como ilustra a Figura 2. Uma aplicação pode ter várias *Activities*, e cada *Activity* pode conter diferentes elementos gráficos, que é o caso de um botão, uma caixa de texto ou uma *label*. A comunicação entre *activities* é feita por *intents*. Principais métodos:

- onCreate() – método para configuração inicial da *Activity*.
- onDestroy() – método para remover a *Activity*
- onResume() - método que é chamado quando a *Activty* estiver visível preparada para obter e processar entrada de dados de utilizador.

Figura 2-Activity Android

Service e *BroadcastReceiver* – permite aplicações executarem algumas atividades em *background* e fornece também funcionalidades adicionais a outros componentes.

ContentProvider é um provedor de conteúdos (Braga,2012), como mostra a Figura 3, e podem ser:

- Partilha de acesso a dados da aplicação por parte de outras aplicações.
- Envio de dados a um *widget* (atalhos que facilitam o acesso a aplicações e ferramentas).
- Sincronização de dados com o servidor implementando a *AbstractThreadedSyncAdapter*.
- Carregamento de dados na interface gráfica usando *CursorLoader*.

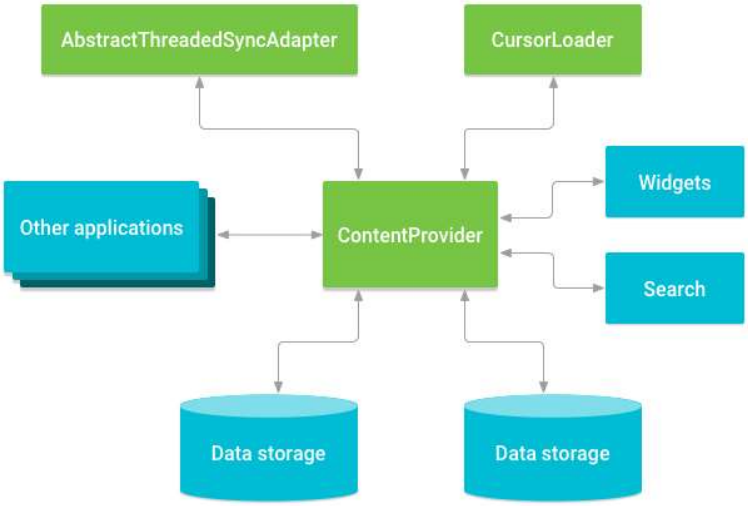

Figura 3-Conteúdo disponibilizados pelo ContentProvider

2.3. Segurança em aplicações móveis

Uma aplicação pode ser desenvolvida com o intuito de ser maliciosa, o que pode levar os atacantes a roubar informação sensível, perda monetária, e até mesmo fazer com que outra aplicação tenha outro tipo de comportamento (Seo et al., 2012).

Mas uma aplicação também pode ser desenvolvida com a preocupação de agradar o utilizador final, sem o querer prejudicar ou sem fins maliciosos. Contudo, essa aplicação pode não seguir boas práticas de desenvolvimento de *software* e segurança, que podem originar a aplicação ter riscos e vulnerabilidades.

Algumas estatísticas comprovam que há muito que aprender e mitigar em relação à segurança em dispositivos móveis, como se pode comprovar a seguir:

- 92% das aplicações com falhas de segurança podem ser exploradas, isto significa que um atacante possivelmente conseguirá com sucesso fazer algum tipo de ataque (Palmer, 2019)
- 19,954 foi o número de vulnerabilidades detetadas em 1,865 aplicações de 259 fornecedores em 2017 (Vijayan, 2019)
- 90% das aplicações móveis foram identificadas com pelo menos uma vulnerabilidade, e essa mesma vulnerabilidade foi explicada no OWASP Mobile Top 10 (Vijayan, 2019)(Huang, 2008)
- Numa investigação feita em 2015, de todas as aplicações móveis testadas quanto a falhas de segurança, 95% eram vulneráveis. Cerca de 35% das aplicações tinham problemas críticos, e 45% das aplicações tinham problemas de segurança de alto risco (Vijayan, 2019)

No desenvolvimento de uma aplicação móvel deve existir cuidados que se não forem tidos em conta podem levar esta a ter vários problemas de segurança, como mostra a Figura 4 (Velu, 2016), onde são exibidos alguns pontos fulcrais na segurança de uma aplicação móvel.

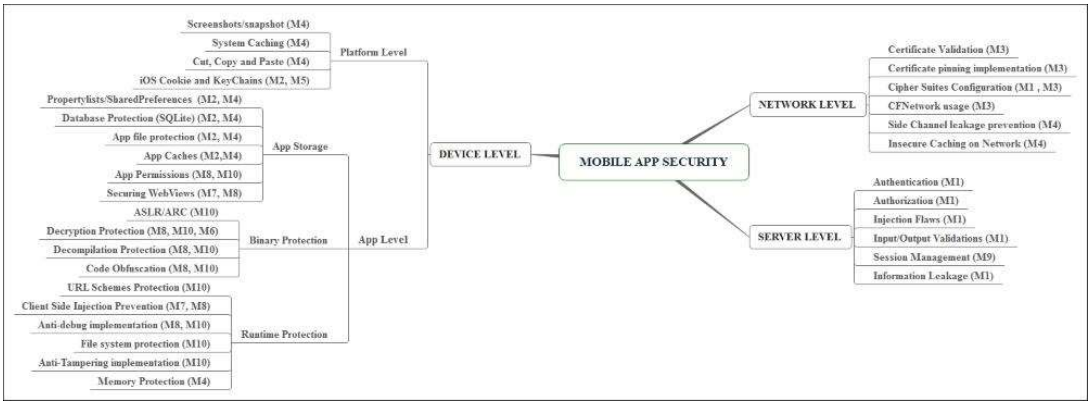

Figura 4-Problemas de segurança das aplicações móveis (Velu 2016)

Problemas de plataforma podem ser considerados de nível mais básico em comparação com os demais, mas são problemas que o programador deve ter bastante atenção. A falta de segurança pode provocar a ação indevida à captura de ecrã, ao acesso a ficheiros temporários e à gestão de *passwords*.

Problemas no armazenamento de dados aparecem quando uma aplicação pretende guardar os seus dados no dispositivo. A aplicação pode guardar dados de três maneiras diferentes: armazenamento interno, armazenamento externo e provedores de conteúdo. Quando a opção é guardar no armazenamento interno, os ficheiros que são criados apenas podem ser acedidos pela aplicação correspondente, esta condição pode assegurar uma certa segurança na maioria das aplicações. No armazenamento externo, como é o exemplo os cartões SD, qualquer informação pode ser acedida por todas as aplicações, por este motivo não se deve guardar informações confidenciais neste local. E por fim, um provedor de conteúdo possui um sistema de armazenamento estruturado que dá a opção de poder ser limitada à aplicação correspondente ou permitir que o acesso a informações da própria aplicação seja feito também por outras aplicações.

Aplicações com falhas de segurança no armazenamento de dados, podem ter problemas na proteção das bases de dados, dos ficheiros da aplicação, dos ficheiros temporários e nas permissões da aplicação.

Problemas na proteção do binário da aplicação é uma grande falha no desenvolvimento da aplicação, pois fará com que existam problemas na criptografia que de uma forma mais fácil para o leitor perceber, seria como misturar todos os dados e transformá-los em algo impercetível para que as informações armazenadas no telefone não possam ser acedidas por outros utilizadores, pode aparecer problemas na descompilação de código e código ofuscado, ou seja, partes do código que fazem certas ações não serem visíveis.

Problemas causados quando a aplicação está a correr podem, por exemplo ser procedidos pela falta de memória alocada, consequentemente fazendo com que apareça falhas na aplicação. Podem aparecer problemas como injeção do lado do cliente, ou seja, injeção de código malicioso no dispositivo móvel através da aplicação, problemas no sistema de ficheiros da aplicação móvel e também falha na proteção da memória.

Problemas ao nível da rede, são problemas com origem nas comunicações entre dispositivos, podendo aparecer falhas na validação dos certificados, fazendo com que exista a possibilidade de um utilizador mal-intencionado intercete a comunicação com o objetivo de alterar ou roubar dados.

Problemas no servidor faz com que apareça falhas, por exemplo, de autenticação e autorização. Um atacante pode ter acesso indevido à aplicação e com privilégios de administrador e pode também haver problemas na gestão de sessões.

2.4. Seguro através do Desenho (*Secure by Design*)

Seguro através do Desenho (*Secure by Design*) é uma abordagem no desenvolvimento de sistemas em segurança (Fernandez, 2004). Tem o objetivo de proporcionar fiabilidade, confidencialidade e integridade nos sistemas a que se propõem com esta abordagem. Nesta abordagem a segurança é pensada e calculada através do primeiro momento antes do desenvolvimento do projeto começar.

São utilizados logo à partida padrões de segurança e outras técnicas que se adequem ao projeto. Estas técnicas e padrões de desenho oferecem uma base de requisitos técnicos e operacionais para proteger o sistema. Algumas técnicas são:

- Construir e manter a segurança do sistema
- Proteção dos dados
- Manter um programa de gestão de vulnerabilidades
- Implementar medidas rigorosas de autenticação e autorização
- Constante monitorização e testes
- Manter uma política de informação para todas as equipas envolventes

Esta abordagem parte do princípio de que problemas de segurança vão acontecer, e para tal são tomadas logo medidas para antecipar os problemas que possam acontecer.

2.5 Seguro por Defeito (*Secure by Default*)

Seguro por Defeito (*Secure by Default*) significa que uma tecnologia ou sistema tem a melhor segurança desenvolvida pelas configurações padrão. Ou seja, isto significa que à partida as

configurações padrão são as configurações mais seguras que se podem ter, isto não significa que sejam as mais simples para utilizador (Stanek, 2017).

Normalmente o nível de segurança é inversamente proporcional à facilidade de utilização.

Esta estratégia significa ter uma abordagem para resolver os problemas de segurança logo no início do projeto, em vez de surgir um problema num estado avançado do sistema e ter que agir em escala para reduzir o impacto causado.

2.6 Ferramentas de análise e deteção de vulnerabilidades

Com todos estes problemas e falhas de segurança encontradas, surgirão vulnerabilidades nas aplicações móveis, que dependendo do tipo poderão ter pouco ou grande impacto ao utilizador. E com a possível falta de conhecimentos teóricos e técnicos sobre estas mesmas vulnerabilidades faz com que os programadores recorram às ferramentas de deteção de vulnerabilidades, que permitem entender se a sua aplicação é segura ou não, e perceber que falhas foram encontradas com o objetivo de melhorar o nível de segurança da mesma.

A seguir vão ser apresentadas algumas ferramentas de análise e deteção de vulnerabilidades, bem como a explicação do seu funcionamento.

As ferramentas abaixo identificadas foram escolhidas por serem ferramentas automáticas de deteção de vulnerabilidades para aplicações *Android*.

2.6.1. Androbugs Framework

Androbugs (Lin, 2015) é uma ferramenta de análise de vulnerabilidades de segurança em aplicações *Android*, que pode ser usada para verificar se contém vulnerabilidades possíveis de serem exploradas por *hackers*, se contém código ofuscado verifica comandos particularmente "perigosos" da *Shell* (por exemplo "*sudo*", comando que dá privilégios administrativos) e também perceber se o código atende às melhores práticas de desenvolvimento.

Yu-Chen Lin autor desta ferramenta, não se preocupou muito com o aspeto visual, mas sim com a maior eficiência possível, sendo que o tempo médio de análise de uma aplicação não deve ultrapassar os 2 minutos.

Quando é concluída a análise da aplicação *Android*, a ferramenta apresenta o resultado ao programador. Este resultado pode ser dividido em 3 secções:

- Na primeira secção contém informações básicas da aplicação analisada. Pode encontrar: qual a plataforma, o nome da pasta, a versão, nome da compilação, versões do SDK (a mínima e a do destino) e por fim todas as assinaturas (MD5, SHA1, SHA256, SHA512)

- Na segunda secção estão colocadas todas as informações quanto ao resultado da busca de vulnerabilidades, dispondo de uma lista de vulnerabilidades e falhas encontradas. Cada uma delas é classificada dependendo do quão grave pode ser. Podem ser classificadas com:
    - Critical – vulnerabilidade que tem obrigatoriamente ser resolvida.
    - Warning – vulnerabilidade que o programador tem manualmente confirmar e corrigir.
    - Notice – vulnerabilidade com severidade baixa.
    - Info – informação, melhorias no código.

  Esta classificação pode ser bastante útil para os programadores, pois poderão reformular o código e priorizar atividades.

  A cada vulnerabilidade encontrada é dada uma descrição do problema e links para possível resolução.

- Na terceira secção contém estatísticas de tempos médios de análise

Com esta ferramenta também se pode fazer uma análise massiva de aplicações, isto é, um grande conjunto de aplicações a serem analisadas ao mesmo tempo.

2.6.2. Droidstatx

Uma ferramenta de segurança para aplicações *Android* que gera um mapa *Xmind* com toda a informação obtida de uma análise estática e identifica possíveis vulnerabilidades na aplicação.

O DroidStatx foi construído por Cláudio André, e o seu objetivo principal que o levou a desenvolver esta ferramenta foi "ajudar programadores que fazem testes de intrusão a cobrir as áreas mais importantes durante uma avaliação." (André, 2019)

Um dos pontos principais desta ferramenta é no mapeamento das vulnerabilidades, pois todas elas são categorizadas segundo o OWASP Mobile Top 10 (consultar a secção 2.7.1.1. OWASP Mobile Top 10).

O resultado final será um ficheiro *Xmind* organizado por categorizas do OWASP Mobile Top 10, contendo vários tópicos e/ou vulnerabilidades, cada vulnerabilidade contém um *link* para o respetivo capítulo no OWASP Mobile Test Guide, explicando a vulnerabilidade, o problema e como confirmá-la.

2.6.3. Androwarn

Ferramenta criada por Thomas Debize em 2013, com o objetivo de detetar e reportar ao programador de possíveis comportamentos maliciosos e vulnerabilidades encontradas numa aplicação *Android*.

O Androwarn (Debize, 2019) faz uma análise de várias categorias, verificando a existência de vulnerabilidades, são elas:

- Extração de identificadores de telefone
- Extração das configurações do dispositivo
- Extração das informações de GPS
- Extração de dados PIM (contactos, calendários, SMS …)
- Interceção de dados de áudio e vídeo
- Execução arbitraria de código
- Negação de serviços

Por fim esta ferramenta apresenta o resultado com todas as informações detetadas assim como as suas vulnerabilidades. Estes resultados podem ser gerados em texto, *HTML* ou *JSON*.

2.6.4. Cuckoo-Droid

O Cuckoo-Droid (I. Revivo and O. Caspi, 2015) é uma extensão do Cuckoo Sandbox (software para automatizar a análise de ficheiros possivelmente maliciosos). Esta ferramenta é usada para executar e analisar aplicações *Android* automaticamente e entregar resultados da busca de vulnerabilidades. Também permite o uso de recursos do Cuckoo Sandbox para analisar *malware Android* por meio de análises estáticas e dinâmicas. A ferramenta é composta por um nó de gestão e vários nós escravos, que podem ser simuladores *Android* ou máquinas virtuais em *Linux* na nuvem.

Esta ferramenta de análise e deteção de vulnerabilidades e *malware* contém um largo número de técnicas de deteção para ocultar o simulador *Android* e fornecer uma análise estática e completa (Wang et al., 2015). Por fim a ferramenta agrega todas as informações que foram encontradas e constrói um relatório pormenorizado para ser entregue ao programador.

2.6.5. Super Android Analyzer

O Super Android Analyzer é uma ferramenta *open source* criada por um grupo de programadores cujo nome é SUPER team, construída em *Rust* e que através da linha de comandos analisa ficheiros com extensão ".apk" e que permite a busca de vulnerabilidades desses mesmos ficheiros. Para tal, a ferramenta descompacta os APKs (aplicações *Android*) e

aplica uma série de regras para detetar essas vulnerabilidades (Pin, Salas, & Eguia, 2018). Contém cerca de 37 regras, que cumpridas geram uma dada vulnerabilidade. É totalmente extensível, visto que todas as regras foram escritas num ficheiro *JSON* e qualquer programador pode facilmente acrescentar outra ou criar a sua própria regra, tornando assim a ferramenta muito mais robusta e sensível a novas ameaças que surgem a toda a hora.

O código foi pensado de maneira que seja possível estar continuamente a ser desenvolvido e com a possibilidade de acrescentar novas funcionalidades.

Esta ferramenta foi construída com o intuito de abranger mais as possibilidades de haver mais ferramentas de análise de aplicações Android e que não tivessem sido escritas em *Java* ou *Python*.

Por fim, o Super Android Analyzer gera um relatório com todos os resultados encontrados (esse mesmo relatório pode ser personalizado), boas práticas de segurança e desenvolvimento do código e que de seguida serão entregues ao programador. Um relatório *HTML* pode ser consultado para tornar a pesquisa de código mais fácil e completa.

2.6.6. JAADAS

JAADAS é o acrónimo para "Joint Advanced Defect Assessment for Android Aplications" e foi desenvolvida em 2014. JAADAS é uma ferramenta escrita com as linguagens de programação *Java* e *Scala* (Shrivastava, A. 2017). Também utiliza a *framework Soot* que tem a função de analisar, instrumentar, otimizar e visualizar aplicações *Java* e *Android* e análise de código de bytes *Java*.

A sua API inclui várias áreas de análise:

- Análise de uso indevido do *SDK*
- *Intent* indevido
- Controle de fluxo entre procedimentos
- Entre outras áreas...

O JAADAS pode ser executado em dois modos:

- Análise rápida
- Análise completa

Quando se utiliza o modo de análise rápido a ferramenta utilizada irá garantir a máxima flexibilidade e reduzir o consumo de tempo neste modo e tem como objetivo realizar uma auditoria básica de segurança.

Na análise completa a ferramenta executa com maior pormenor uma exaustiva busca de vulnerabilidades à aplicação *Android*, incluindo um fluxo de dados completo e análise de

procedimentos. Este modo consome muitos recursos comparativamente ao modo de análise rápida, portanto, na análise completa o tempo de processamento irá ser bastante superior.

2.6.7 Mobile Security Framework

Mobile Security Framework (MobSF) é uma *framework open source* de *pentesting* automatizada para aplicações móveis em *Python*. Desenvolvida por Ajin Abraham, Magaofei, Matan Dobrushin e Vincent Nadal.

MobSF pode ser executada nos sistemas operativos *Android*, *iOS* e *Windows*, capaz de realizar análises estáticas e dinâmicas. Na análise estática, a aplicação é testada de dentro para fora, ou seja, analisa o código-fonte ou binário sem executar a aplicação. Por fim é armazenado todas as vulnerabilidades para posteriormente serem verificadas pelo programador. Numa análise dinâmica esta *framework* ajuda a realizar uma avaliação de segurança em tempo de execução.

O objetivo desta *framework* é ajudar os programadores a desenvolver aplicações móveis seguras e identificar vulnerabilidades em todos os momentos de desenvolvimento.

2.7. Iniciativas na área da segurança de aplicações móveis

Nesta secção irão ser descritos alguns documentos e projetos centrados na área da segurança de aplicações móveis.

O objetivo principal dos documentos e projetos abaixo apresentados é educar e consciencializar programadores, gestores e organizações sobre as consequências das mais importantes vulnerabilidades de segurança de aplicações móveis. Estes fornecem técnicas básicas de segurança, o tipo de falhas encontradas, como corrigir e mitigar as mesmas e por fim dar o conhecimento necessário de como se proteger contra algumas vulnerabilidades de alto risco na segurança de informação.

2.7.1 Recomendações, Guias e Normas

Neste capítulo são apresentados algumas recomendações, guias e normas na área da segurança das aplicações móveis.

2.7.1.1 OWASP Mobile Top 10

O OWASP Mobile Top 10 (OWASP, 2016) é um documento produzido pelo OWASP, mas com o objetivo de fornecer aos programadores e equipas de segurança os recursos necessários para criar e manter aplicações móveis seguras e incorporar as melhores práticas de programação.

É um documento que destaca falhas e vulnerabilidades de segurança em aplicações móveis. Tal como o projeto referido em cima, este também se divide em 10 categorias:

- M1 – Utilização impropria da plataforma - Esta categoria centra-se no uso incorreto da plataforma ou a não utilização dos controles de segurança
- M2 – Armazenamento inseguro de dados - Esta categoria abrange vulnerabilidades ou riscos se alguma informação confidencial for armazenada pela aplicação e se essa informação foi ou não protegida
- M3 – Comunicação Insegura – Aqui entram vulnerabilidades referentes a ligações à internet inseguras
- M4 – Autenticação insegura - Esta categoria inclui problemas de gestão de sessões, de autenticação
- M5 – Criptografia insuficiente – Podem entrar nesta categoria vulnerabilidades referentes a algoritmos de criptografia fracos ou a falta deles
- M6 – Autorização insegura - Esta categoria centra-se nas falhas de autorização
- M7 – Má qualidade do código – São categorizadas vulnerabilidades referentes a mau desenvolvimento de código
- M8 – Violação do código – Entram aqui vulnerabilidades que dão o acesso ao código fonte
- M9 – Engenharia reversa - Qualquer aplicação que seja feita engenharia reversa, entra nesta categoria
- M10 – Funcionalidade indesejada – Vulnerabilidades que ofuscam atividades nas aplicações

2.7.1.2 Vetting the Security of Mobile Applications

"*Vetting the Security of Mobile Applications*" (Ogata et al, 2019) é um documento que descreve um processo para aplicações móveis. Esse processo tem como objetivo garantir que aplicações móveis tenham um nível de segurança alto e que estão em conformidade com certos requisitos de segurança e que por fim fiquem livres de vulnerabilidades.

O processo possui 4 etapas:

- Pré-processamento – Na primeira etapa a aplicação é recebida para análise, de seguida são recolhidas algumas informações sobre a mesma, depois é descompilado o código.
- Teste – Na segunda etapa são executadas algumas ferramentas de análise estática, para verificar a existência de vulnerabilidades, e por fim é gerado um relatório dos riscos associados.
- Verificação – Na terceira etapa, um analista de segurança inspeciona todos os dados e relatórios associados para no fim determinar se a aplicação viola as políticas de segurança.
- Submissão – Na quarta etapa são submetidos todos os resultados, o relatório de análise feita e se a aplicação é ou não aprovada.

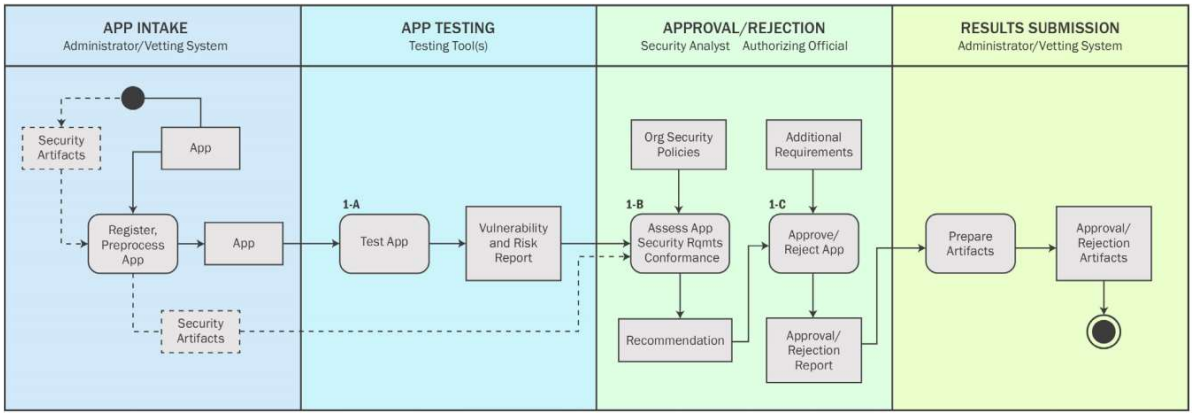

Figura 5-As 4 etapas do processo de verificação

2.7.1.3 NIAP

O NIAP (National Information Assurance Partnership) (NIAP,2020) é um documento desenvolvido para atender às necessidades de empresas, profissionais de segurança de informação e programadores na realização de testes de segurança. Este documento foi desenvolvido pela National Security Agency (NSA) pela National Institute of Standarts and Technology (NIST).

O NIAP utiliza um conjunto de requisitos e objetivos de segurança, para implementação numa certa categoria de tecnologia de informação para atender às necessidades dos clientes. Esses requisitos.

Os requisitos definidos neste documento podem ser divididos em duas categorias:

- Requisitos funcionais – Características sobre os atributos do software, alguns requisitos são apresentados na Tabela 1.
- Requisitos de garantia – Medidas que se devem tomar para que a verificação seja executada com sucesso.

Functional Requirements
Access to Platform Resources
Anti-Exploitation Capabilities
Cryptographic Key Functionality
Cryptographic Operations
Encryption of Sensitive Application Data
Hyper Text Transfer Protocol Secure (HTTPS)
Integrity for Installation and Update
Network Communications
Protection of Data in Transit
Random Bit Generation
Secure by Default Configuration
Software Identification and Versions
Specification of Management Functions
Storage of Credentials
Supported Configuration Mechanism
Transport Layer Security Operations
Use of Supported Services and Application Programming Interfaces
Use of Third-Party Libraries
User Consent for Transmission of Personally Identifiable Information
X.509 Functionality

Tabela 1-Requisitos Funcionais

2.7.2 Listas de Vulnerabilidades

Nesta secção serão exibidas algumas listas de vulnerabilidades e falhas que foram colecionadas, de modo a dar ao programador a noção do impacto de cada uma delas.

2.7.2.1 NVD

O NVD (National Vulnerability Database) é uma base de dados de vulnerabilidades encontradas e relatadas por programadores.

Cada vulnerabilidade identificada possui um conjunto de elementos, são eles:

- CVE (explicado na secção 2.7.2.2).
- Uma descrição sobre a vulnerabilidade em questão.
- CVSS (Commom Vulnerability Scoring System), ou seja, uma pontuação é dada à vulnerabilidade dependendo do seu nível de severidade, quanto maior a pontuação recebida maior será o perigo dessa mesma vulnerabilidade.
- Referencias sobre ameaças, ferramentas e soluções a ter para resolver esta ameaça

Esta base de dados inclui ainda falhas de software relacionadas com a segurança, configurações incorretas e outras métricas de impacto.

2.7.2.2 Android - CVE

O CVE (Common Vulnerabilities and Exposures) para *Android* é uma lista de vulnerabilidades e falhas que podem ser encontradas em dispositivos *Android*. Cada CVE tem um identificador que identifica uma certa vulnerabilidade, uma pequena descrição e algumas referências, por exemplo, relatórios ou avisos sobre a vulnerabilidade.

Esta lista de vulnerabilidades funciona como um dicionário, diferente de uma base de dados, que contém um aglomerado de informações. Neste caso, cada CVE possui um *link* direcionado para um NDV (referido na secção 2.7.2.1), contendo um conjunto de informações mais detalhadas à cerca da vulnerabilidade correspondente.

Os CVEs ajudam os profissionais de segurança e os programadores a perceber qual a ameaça detetada e priorizar os seus esforços para solucionar o problema.

2.8. Uniformização de vulnerabilidades

Todos os projetos, documentos e listas de vulnerabilidades que foram apresentadas, identificam as vulnerabilidades de forma e nomes diferentes, consoante as suas normas. Portanto, a mesma vulnerabilidade pode possuir nomes diferentes, o que pode originar confusão.

Para tal surge esta secção, tem como objetivo apresentar algumas técnicas de uniformização das vulnerabilidades, utilizando algoritmos de similaridade de texto.

2.8.1. Jaccard Similarity

Este é um algoritmo de comparação de padrões (Huang, 2008). Compara dois conjuntos e verifica quais palavras são partilhadas pelos dois conjuntos e quais as que são diferentes. No final do resultado em percentagem de 0% a 100%, quanto maior a percentagem mais semelhanças terão os dois conjuntos.

As vantagens deste algoritmo são:

- Simplicidade conceptual
- Simplicidade computacional
- Larga aplicabilidade

A desvantagem deste algoritmo é que á medida que o tamanho de um conjunto aumenta, o número de semelhanças tende a aumentar.

A fórmula para o cálculo de Jaccard Similarity:

$JS(X,Y)=|X \cap Y|/|XUY|$

Um exemplo simples do funcionamento seria:

$A = \{1,2,3,6,8,9\}$

$B = \{0,1,4,5,6,7,8,9\}$

$JS(A,B) = |A \cap B|/|AUB| = \{1,6,8\}/\{0,1,2,3,4,5,6,7,8,9\} = 3/9 = 0,33$

Jaccard Similarity (A,B) = 33%

Com este resultado conclui-se que o conjunto A tem 33% de semelhanças em relação ao conjunto B.

2.8.2. Cosine Similarity

Cosine similarity é um algoritmo aplicado a medições de similaridade de documentos de texto.

“Quando documentos são representados como vetores, a similaridade dos dois documentos corresponde à correlação entre vetores. Isto é quantificado como o cosseno do ângulo entre vetores, ou seja, a similaridade de cosseno”.(Huang, 2008)

As vantagens de utilizar cosine similarity é que mesmo que os tamanhos dos documentos sejam díspares, existe ainda a probabilidade de eles serem semelhantes.

A fórmula de cálculo de Cosine Similarity seria:

$$CS = \cos(\alpha) = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}| \times |\vec{B}|}$$

O cosseno de 0° é 1, isto indica que os dois documentos são idênticos. O cosseno de 90° é 0, isto indica que os dois documentos não têm semelhanças nenhumas. Quanto menor o ângulo, maior serão as suas semelhanças.

2.8.3. Latent Semantic Indexing

Latent Semantic Indexing (Dumais, 2005) é uma técnica de processamento de linguagem natural, que analisa relações entre um conjunto de documentos e as palavras que eles contêm, produzindo um conjunto de conceitos relacionados entre eles.

Este algoritmo assume que palavras com significado próximo ocorrerão em partes semelhantes do texto. É produzida uma matriz contendo a quantidade de vezes que ocorre cada palavra por documento.

De seguida são comparados os documentos, verificando o cosseno do ângulo entre dois vetores. Os valores resultantes que estejam próximos de 1 significa que os documentos são bastante semelhantes, enquanto valores próximos de 0 significa que os documentos são muito diferentes.

2.8.4. Siamese Manhattan LSTM

São redes que possuem duas ou mais sub-redes idênticas. Têm um bom desempenho e são usadas para tarefas como similaridade semântica de frases. Este algoritmo refere-se apenas ao facto de escolher a distância de Manhattan para comparar os estados (N. Othman, 2019).

2.9. Trabalhos relacionados

Existem alguns trabalhos que se enquadram com esta dissertação, de seguida serão abordados alguns exemplos.

1. Este primeiro trabalho utiliza uma técnica de análise de programa específica do *Android* capaz de gerar um grande número de casos de teste para testar a aplicação. Considerando os casos de teste gerados, o sistema executa-os paralelamente em vários emuladores com plataforma *Android* na *cloud*, por fim é gerado um relatório de falhas.(Malek et al., 2012)

2. O AppInspector é um sistema automatizado de teste e validação de segurança. Utiliza uma abordagem dinâmica que monitoriza o uso de informações confidenciais de uma aplicação e verifica comportamentos suspeitos como consumo excessivo de recursos ou o acesso de dados do utilizador. Esta análise é executada na *cloud*.(Gilbert, Chun, Cox, & Jung, 2011)

3. Este sistema utiliza uma abordagem para análise de segurança dinâmica automática de aplicações *Android*. O AppsPlayground integra um conjunto de ferramentas para a deteção de *malware* nas aplicações. (Rastogi, Chen, & Enck, 2013).

2.10. Conclusões

Apesar de haver um grande número de problemas na segurança nas aplicações móveis (referidos no ponto 2.3), também se pode concluir, através das secções apresentadas anteriormente, que existe consciência por parte de organizações e empresas no que toca no desenvolvimento de conteúdos relativos à segurança das aplicações móveis.

Esses conteúdos que podem ser documentos, lista de vulnerabilidades ou até ferramentas de análise de vulnerabilidades, têm o objetivo de ajudar o programador no desenvolvimento de aplicações móveis seguras.

Relativamente às ferramentas de deteção de vulnerabilidades, existe uma gama diferente de opções *open source* que detetam vulnerabilidades e *malware*. Logo, é o programador que deve investir na utilização destas ferramentas para desenvolver aplicações móveis em segurança.

CAPÍTULO 3

Desenho e Implementação

Neste capítulo vai ser explicado todo o desenho e implementação do sistema assim como a solução para o problema desta dissertação.

3.1. Introdução

Esta secção descreve a arquitetura e implementação do sistema de identificação de vulnerabilidades e entrega de relatório para os programadores.

O sistema foi desenvolvido para integrar recursos de identificação de vulnerabilidades em aplicações *Android* (APKs) e fornecer um relatório necessário para os programadores sobre como mitigar ou eliminar tais vulnerabilidades usando informações de diferentes fontes, ou seja, uma base de dados de material educacional sobre as próprias vulnerabilidades e que deve estar em constante atualização.

O sistema desenvolvido pode-se dividir em dois principais componentes. O primeiro componente do sistema é uma API que pode ser usada para alimentar o sistema com novos APKs que requerem revisão de vulnerabilidades de segurança e também para obter feedback para os programadores sobre os APKs analisados. O segundo componente do sistema é uma coleção de componentes de software que serão executados para realizar uma análise de segurança em APKs *Android*, e por fim fornecer um relatório detalhado aos programadores.

O principal objetivo do desenvolvimento deste sistema é prevenir que aplicações mal desenvolvidas, com falhas de segurança ou mal concebidas fiquem disponíveis numa *App Store* para o utilizador final. E também ser capaz de lidar com a grande quantidade de APKs que todos os dias são enviados e colocados na *App Store*.

O funcionamento deste sistema deve ser capaz de testar cada uma das novas aplicações enviadas, correr um conjunto de ferramentas e determinar a existência de vulnerabilidades de segurança (classificadas de acordo com o OWASP Mobile Top 10) e retornar um relatório combinado com tudo o for encontrado sobre as vulnerabilidades aos programadores. Este relatório é educacional e devidamente construído para que seja facilmente possível detetar quais as medidas apropriadas para corrigir ou mitigar as vulnerabilidades encontradas.

3.2. Arquitetura do sistema e componentes principais

Nesta secção vai ser apresentada a arquitetura do sistema bem como os seus principais componentes. Para tal, o sistema começa por receber uma aplicação (APK) por fontes externas (programadores ou *App Stores*), de seguida realiza alguns testes a essa APK e por fim fornece um relatório detalhado sobre as descobertas relacionadas à segurança do APK para os programadores.

As funcionalidades existentes são expostas a clientes externos por meio de uma API REST que isola completamente os processos de troca de informações das operações internas. A Figura 6 ilustra o sistema desenvolvido.

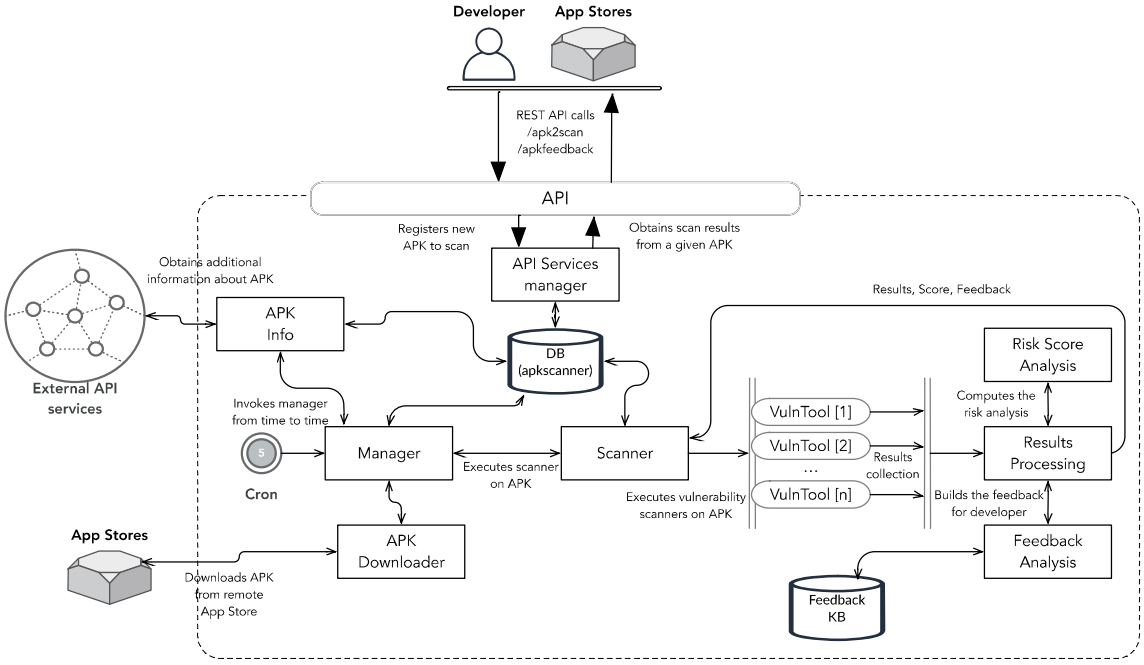

Figura 6-Arquitetura geral da Análise de Vulnerabilidades e Sistema de Feedback (Fonte projeto AppSentinel)

3.2.1 Armazenamento de informação

O sistema armazena todas as informações necessárias e os resultados do processo de análise de vulnerabilidades numa base de dados MySQL. A base de dados contém as seguintes informações:

- "**`apk`**": contém todas as informações sobre um APK descarregado, ou seja, todas as informações de metadados que existem sobre o APK que está sendo analisado pelo sistema.
- **"`apk2scan`"**: contém as informações sobre os APKs que foram enviados ao sistema, mas ainda não foram processados.

- **"`apkresults`"**: contém informações sobre os resultados obtidos após a análise das ferramentas à referida APK.

Estas tabelas agregam todas as informações necessárias que o sistema usa para analisar os APKs enviados e armazena informações sobre os resultados da análise (vulnerabilidades, relatório aos programadores e um valor de risco de segurança).

3.2.2 API

A API permite que sistemas externos interajam com as operações internas do sistema. Oferece as funcionalidades de análise de vulnerabilidades de APKs, bem como o fornecimento dos resultados da APK enviada para análise e relatório detalhado dos APKs analisados.

Apkscan – Esta chamada à API é usada pelo cliente para enviar uma nova aplicação ao sistema para ser analisada. O cliente deve enviar (POST) um identificador exclusivo MD5 (md5) da aplicação para verificar. O MD5 é utilizado pela app store para identificar exclusivamente as diferentes aplicações enviadas. Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação POST `/apkscan`, passando o identificador da aplicação para verificação (md5);
2. O servidor API recebe a solicitação, verifica a existência desse MD5;
3. O servidor API armazena o MD5 numa base de dados (`apkscanner`);
4. O servidor da API retorna o resultado da operação ao cliente.

O resultado da operação é recebido como uma mensagem no formato *JSON* que possui a seguinte estrutura:

- "**`status`**": uma resposta booleana, que pode ser "verdadeira" se o MD5 estiver correto ou "falsa" se o mesmo estiver errado.

- "**`message`**": uma mensagem informativa dependente do status.

Apkfeedback – Esta chamada à API é usada pelo cliente para solicitar o relatório elaborado com todas as informações à análise à aplicação proposta, enviada anteriormente ao sistema para processamento. O cliente deve solicitar (GET) os resultados colocando o MD5 (md5) da aplicação para o qual ele solicita o relatório. Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação GET para a chamada da API `/apkfeedback`, passando o identificador da aplicação para verificação (md5);
2. O servidor da API recebe a solicitação, verifica se existe esse MD5;
3. O servidor da API solicita à base de dados (`apkscanner`) os resultados existentes para o MD5 pedido;
4. O servidor da API retorna os resultados ao cliente.

A resposta para esta solicitação é o problema que se está a tentar resolver nesta dissertação e que é uma mensagem no formato *JSON* e tem a seguinte estrutura, dependendo da situação:

• Se ocorreu um erro na especificação do MD5 ou se a aplicação solicitada ainda não foi processada, é recebida uma mensagem no formato *JSON* contendo um campo com o nome "**`status`**" indicando o sucesso ou falha da chamada da API e um campo "**`message`**" com alguns detalhes adicionais do resultado da operação.

• Se a aplicação já tiver sido processada pelo sistema e os resultados já estiverem disponíveis, será recebida uma mensagem no formato *JSON* contendo um campo de "**`status`**", um campo "**`results_history`**" com informações sobre o histórico da análise desta aplicação ao longo do tempo, isto se a aplicação já tiver sido processada mais que uma vez, e um campo "**`results`**" com os resultados da análise da aplicação e algumas informações adicionais.

Apkvullevel – Esta chamada à API é utilizada pelo cliente para receber um relatório da análise à aplicação requerida, apenas com os nomes de cada vulnerabilidade detetada juntamente com o seu nível de severidade. O cliente deve solicitar (GET) os resultados colocando o MD5 (md5) da aplicação para o qual ele solicita o relatório. Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação GET para a chamada da API `/vulnerabilities`, passando o identificador da aplicação para verificação (md5);
2. O servidor da API recebe a solicitação, verifica se existe esse MD5;
3. O servidor da API solicita à base de dados (`apkscanner`) os resultados existentes para o MD5 pedido;
4. O servidor da API retorna os resultados ao cliente.

O resultado à chamada desta API é uma resposta no formato *JSON*, com a seguinte estrutura:

- "**`status`**": retorna verdadeiro se o MD5 existir e retorna falso se não encontrar o MD5 ou houver algum problema na construção do relatório.
- "**`vulnerabilities`**": é apresentada uma lista com vários objetos *JSON*:
  - "**`vulnerability`**": é identificada o nome da vulnerabilidade;
  - "**`severity`**": o nível de severidade;

Apkmonthlevels – Ao solicitar esta chamada à API, o utilizador irá receber um relatório organizado por meses, com a quantidade de vulnerabilidades que foram detetadas por nível de severidade. O cliente deve solicitar (GET) os resultados colocando o id (o número de meses). Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação GET para a chamada da API / `vulnerabilities`, passando um id de número de meses, este id varia entre 1 a 12;
2. O servidor da API recebe a solicitação, e faz a contagem de todas as aplicações analisadas dos últimos x (id) meses;
3. O servidor da API retorna os resultados ao cliente;

O resultado desta operação é uma mensagem no formato *JSON* que possui a seguinte estrutura:

- "**`status`**": retorna verdadeiro se existirem aplicações analisadas e falso se houver algum problema na construção do relatório;
- "**`info`**": uma lista de todos os meses com os respetivos números de apks por severidade
    - "**`month`**": nome do mês;
    - "**`notice`**": número de vulnerabilidades analisadas no mês respetivo, cujo nível de severidade foi notice;
    - "**`warning`**": número de vulnerabilidades analisadas no mês respetivo, cujo nível de severidade foi warning;
    - "**`critical`**": número de vulnerabilidades analisadas no mês respetivo, cujo nível de severidade foi critical;

Apklevels – Esta chamada à API é utilizada pelo cliente para solicitar o score de uma dada aplicação, ou seja, esse score varia entre 0 e 1, quanto maior o seu valor, mais perigosa será a aplicação. O cliente deve solicitar (GET) os resultados colocando o MD5 (md5) da aplicação para o qual ele solicita o score. Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação GET para a chamada da API `/vulnerabilities/levels`, passando o identificador da aplicação para verificação (md5);
2. O servidor da API recebe a solicitação, verifica se existe esse MD5;
3. O servidor da API solicita os dados e constrói a resposta.
4. O servidor da API retorna os resultados ao cliente.

O resultado para esta solicitação é uma resposta *JSON* que possui a seguinte estrutura:

- Se ocorreu um erro na especificação do MD5 ou se a aplicação solicitada ainda não foi processada, é recebida uma mensagem no formato *JSON* contendo um campo com o nome "**`status`**" indicando o sucesso ou falha da chamada da API e um campo "**`message`**" com alguns detalhes adicionais do resultado da operação;
- Um valor decimal num campo com o nome "score" que varia entre 0 e 1 e mede o nível de risco que a aplicação possui;

Allapkvulnlevels – A diferença entre esta chamada à API e a apkmonthlevels, é que a apkmonthlevels apresenta o número de vulnerabilidades por nível de severidade de aplicações analisadas por cada mês, enquanto o allapkvullevels apresenta o número de vulnerabilidades por

nível de severidade de todas as aplicações analisadas. Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação GET para a chamada da API `/vulnerabilities/levels`;
2. O servidor da API recebe a solicitação, verifica se existe MD5 na base de dados;
3. O servidor da API solicita os dados.
4. O servidor da API retorna os resultados ao cliente.

O resultado para esta solicitação é uma resposta *JSON* que possui a seguinte estrutura:

- "**`status`**": se houver aplicações analisadas retorna uma resposta verdadeira, se não houver retorna uma resposta falsa;
- "**`notice`**": número de vulnerabilidades analisadas no mês respetivo cujo nível de severidade foi notice;
- "**`warning`**": número de vulnerabilidades analisadas no mês respetivo cujo nível de severidade foi warning;
- "**`critical`**": número de vulnerabilidades analisadas no mês respetivo cujo nível de severidade foi critical;

Apkslist – Ao solicitar esta chamada à API, o utilizador irá receber um relatório com a quantidade de vulnerabilidades que foram detetadas por nível de severidade por cada aplicação analisada. O cliente deve solicitar (GET) os resultados chamando a API. Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação GET para a chamada da API `/vulnerabilities/apks/list`;
2. O servidor da API recebe a solicitação;
3. O servidor da API retorna os resultados ao cliente;

O resultado desta operação é uma mensagem no formato *JSON* que possui a seguinte estrutura:

- "**`status`**": retorna verdadeiro se existirem aplicações analisadas e falso se houver algum problema na construção do relatório;
- "**`list`**": uma lista de objetos *JSON*, contendo os seguintes campos:
  - "**`apk_md5`**": neste campo é apresentado o MD5 da aplicação.
  - "**`vulnerability_levels`**": uma lista com o número de vulnerabilidades por cada nível de severidades. Para cada nível de severidade notice, warning e critical será apresentado o número de vulnerabilidades existentes.
- "**`downloads`**": neste campo é apresentado o número de downloads feitos na *App Store*.
- "**`rating`**": valor que varia entre 0 e 5 avaliando a aplicação.

ShowKnowledgeBase – Esta chamada à API é utilizada pelo cliente para solicitar todos os conteúdos existentes na base de conhecimentos (explicado na secção 3.3.7). O cliente deve solicitar (GET) os resultados chamando a API. Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação GET para a chamada da API `/knowledgeBase/content`;
2. O servidor da API recebe a solicitação;
3. O servidor da API solicita os dados.
4. O servidor da API retorna os resultados ao cliente.

O resultado para esta solicitação é uma resposta *JSON* que possui a seguinte estrutura:

- "**`results`**": é apresentada uma lista de objetos *JSON* dispondo todas as informações colecionadas na base de conhecimentos, o formato do objeto é:
    - "**`category`**": neste campo é apresentado a que categoria do OWASP Mobile Top 10 a vulnerabilidade em questão.
    - "**`name`**": algumas palavras chave que identificam a vulnerabilidade.
    - "**`links`**": uma lista de links de apoio educacional para ajudar o programador a consciencializar-se à cerca da vulnerabilidade.
    - "**`books`**": uma lista de livros de apoio educacional para ajudar o programador a consciencializar-se à cerca da vulnerabilidade.
    - "**`articles`**": uma lista de artigos de apoio educacional para ajudar o programador a consciencializar-se à cerca da vulnerabilidade.

PostKnowledgeBase - Ao solicitar esta chamada à API, o utilizador irá adicionar informações à base de conhecimentos (explicado na secção 3.3.7). Ao adicionar mais informações faz com que a base de conhecimentos consiga abranger todo o tipo de vulnerabilidades. O cliente deve enviar (POST) as palavras chaves que identificam a vulnerabilidade, de seguida adicionar as informações que pretende adicionar á base de conhecimentos *links*, livros ou artigos. Esta chamada da API funciona da seguinte maneira:

1. O cliente envia uma solicitação POST / `knowledgeBase/add`, passando as palavras chave que identificam a vulnerabilidade à qual pretende adicionar informações e os respetivos links, livros e artigos;
2. O servidor API recebe a solicitação, verifica a existência dessas palavras chave;
3. O servidor da API retorna o resultado da operação ao cliente.

3.3. Processamento das aplicações Android

Nesta secção será explicado toda a preparação, processamento, análise, categorização das vulnerabilidades, uniformização das vulnerabilidades e entrega de relatório, quando uma aplicação *Android* é fornecida ao sistema.

3.3.1 Estrutura

O sistema tem como objetivo ajudar os programadores no desenvolvimento de aplicações móveis, analisando-as e por fim entregar um documento com todas as vulnerabilidades encontradas e informações acerca das mesmas.

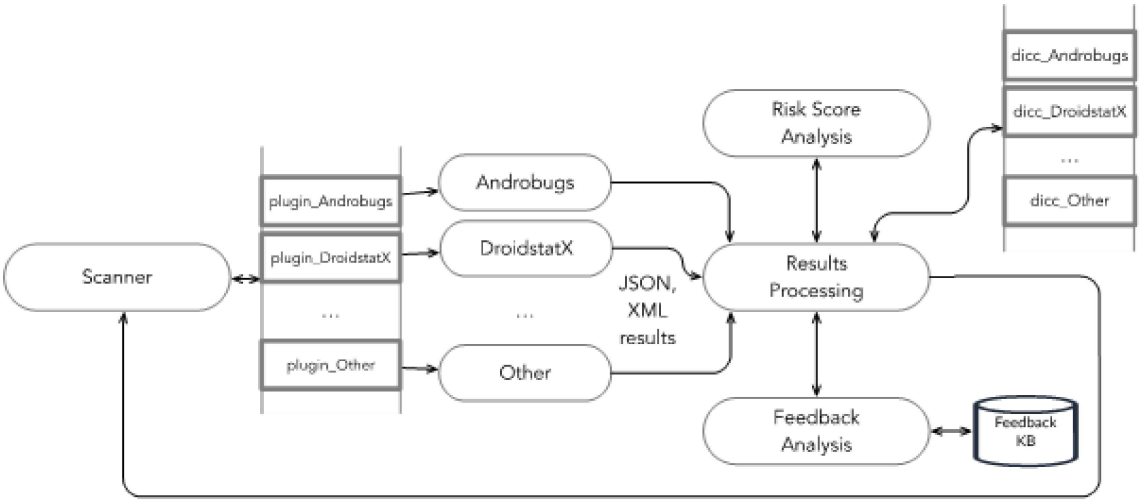

Figura 7-Arquitetura do plugin de análise de vulnerabilidades de segurança no sistema (Fonte projeto AppSentinel)

3.3.2 Manager

O processamento da aplicação é feito pelo *script* "**`manager`**" que é responsável pela operação de correr todas as ferramentas de análise disponíveis. Uma vez que o "**`manager`**" é chamado, ele verifica na base de dados "**`apkscanner`**" se existem aplicações para testar (aplicações que foram adicionadas pela chamada "**`apkscan`**" à API explicada na secção 3.2.2). Ao verificar a base de dados e se confirmar a existência de aplicações para serem testadas, o "**`manager`**" irá recolher uma série de informações à cerca da aplicação. As informações recolhidas da aplicação são:

- O nome da aplicação.
- O nome do pacote da aplicação.
- A versão da aplicação.
- O MD5 (identificador) da aplicação.
- A localização dos ficheiros da aplicação.

De seguida, o "**manager**" irá correr o método "**download_apk**" na qual se conectará à API da *App Store* (Aptoide), faz o download e guarda a aplicação e os seus metadados (informações à cerca da aplicação).

Depois de transferida aplicação é adicionada à base de dados "**apkscanner**" todas as informações recolhidas sobre a aplicação.

Logo a seguir, o "**manager**" chama o *script* "**scanner**" que vai verificar se existem ferramentas de análise disponíveis e integradas no sistema para fazerem a análise às aplicações transferidas.

De seguida é removida da base de dados a aplicação que tinha sido adicionada através da chamada à API "**apkscan**".

3.3.3 Scanner

O desenvolvimento desta arquitetura tem como base um sistema de *plugins*, que consiste em adicionar ou retirar um plugin sem que o funcionamento deste esteja em causa.

Este sistema tem como objetivo a integração de várias ferramentas, com o intuito de poder dar resposta a falsos-positivos e vice-versa, por exemplo, se em 5 ferramentas apenas uma detetou uma certa vulnerabilidade, existe a probabilidade daquela vulnerabilidade ser um falso-positivo, ainda assim o programador deve sempre confirmar manualmente se existe ou não aquele problema na sua aplicação.

O *script* "**scanner**" começa por identificar se existem plugins no *file system*. Depois de detetados, é verificado a cada plugin desenvolvido se está pronto para ser executado através do atributo "`enable`", o seu valor estará a "`true`" se estiver disponível para ser executado e "`false`" se não estiver disponível para ser executado.

Por fim, o "**scanner**" corre os plugins que estiverem habilitados para serem executados.

3.3.4 Plugin

Um plugin é um *script* desenvolvido para executar uma ferramenta de análise associada. Para cada ferramenta de análise de vulnerabilidades disponível no sistema, um plugin executará a ferramenta respetiva, sendo esta solução um sistema de plugins.

Pode adicionar ou remover plugins sem que isto interfira no bom funcionamento da aplicação.

Um plugin começa por executar uma ferramenta de análise. Depois da ferramenta terminar a sua análise à aplicação, são recolhidos os dados gerados pela ferramenta e é construído um relatório com todas as informações fornecidas num formato *JSON*.

A estrutura do relatório é a seguinte:

- "**`vulnerability`**" - indica o nome da vulnerabilidade.
- "**`details`**" – aqui é apresentado informações mais detalhadas à cerda da vulnerabilidade.
- "**`severity`**" - indica o nível de severidade que a vulnerabilidade pode ter. As vulnerabilidades de nível mais baixo apresentam o nível "notice", o segundo nível "warning" e por último, o nível mais elevado "critical". Estes níveis permitem que os programadores entendam rapidamente os riscos de segurança das suas aplicações *Android* e aprendam a tomar as devidas ações para mitigá-los. Quanto maior é o nível de severidade maior é a probabilidade de um atacante poder explorar essa vulnerabilidade e conseguir obter várias informações sobre o utilizador.
- "**`detectedby`**" – indica quais as ferramentas que detetaram aquele problema de segurança.
- "**`feedback`**" – um dos principais objetivos deste sistema é fornecer um relatório apropriado aos programadores, fornecendo informações não apenas sobre as vulnerabilidades de segurança existentes nas aplicações *Android*, mas também informações sobre o modo como as vulnerabilidades detetadas operam e como eles afetam a segurança dos usuários finais e como eles podem corrigi-los ou mitigá-los. Portanto, esse mecanismo visa educar não apenas sobre riscos de desenvolvimento, mas também sobre boas práticas de desenvolvimento que precisam ser empregues para corrigir esses riscos de segurança de desenvolvimento. Aqui serão apresentados todos os conteúdos reunidos para mostrar ao programador como resolver e aprender sobre a vulnerabilidade em si.
    - "**`url`**" – links que levam a vários recursos online sobre a vulnerabilidade.
    - "**`vídeo`**" – links que direcionam para alguns vídeos que contém informação relevante sobre a vulnerabilidade.
    - "**`book`**" – livros sobre a vulnerabilidade.
    - "**`other`**" – qualquer outro conteúdo à cerca da vulnerabilidade, como artigos científicos, jornais, etc…

Para este sistema foram desenvolvidos os seguintes plugins:

Ferramenta	Plugin
Androbugs	Plugin_Androbugs
Droidstatx	Plugin_Droidstatx
Super	Plugin_Super

Tabela 2-Ferramentas e seus plugins respetivos

3.3.5 OwaspEngine

Depois de finalizado a execução de todas as ferramentas e todas elas terem gerado os relatórios à cerca das vulnerabilidades encontradas, entra-se num processo de categorização.

Este processo é executado pelo "**`owaspEngine`**", um *script* que vai categorizar todas as vulnerabilidades detetadas por todas as ferramentas de análise segundo o OWASP Mobile Top 10. Para tal, a solução encontrada foi construir um dicionário de vulnerabilidades referente a cada ferramenta de análise. Para esta dissertação, estão a ser utilizadas três ferramentas de deteção de vulnerabilidades, Androbugs, DroidstatX e Super. E para cada ferramenta existe um dicionário associado:

Ferramenta	Dicionário
Androbugs	Androbugs_dict
Droidstatx	Droidstatx_dict
Super	Super_dict

Tabela 3-Ferramenta e respetivo dicionário

Cada dicionário está num formato *JSON*, é construído antecipadamente e contém todas as vulnerabilidades que a ferramenta em questão pode detetar. O dicionário é estruturado da seguinte maneira:

- "**`name`**" - nome da vulnerabilidade descrita pela ferramenta em causa
- "**`category`**" - indica a que categoria (segundo o OWASP Mobile Top 10) a vulnerabilidade se encontra (pode ir de M1 a M10)
- "**`level`**" - indica o nível de severidade da vulnerabilidade
- "**`keywords`**" – palavras chave associadas à vulnerabilidade em causa, que vai ajudar na normalização das vulnerabilidades detetadas por várias ferramentas e na associação de material educacional para ajuda ao programador.

```
{
    "name": "WebView with Javascript enabled.",
    "category": "M1",
    "level": "Warning",
    "keywords":["webview", "JSenabled"]
},
```

Figura 8-Estrutura de um dicionário

Para categorizar todas as vulnerabilidades expostas nos dicionários foi preciso um estudo minucioso, em que se verifica: o que é a vulnerabilidade, possíveis ataques e o estudo da possível categoria em que se pode encaixar. Este estudo deve ser feito para todas as vulnerabilidades que uma ferramenta pode detetar.

Na categorização das vulnerabilidades, para cada uma delas é verificada o seu contexto, por exemplo, o tipo de ataque, o que é atacado e em quais circunstâncias acontecem o ataque, e por fim, cada vulnerabilidade pode ser comparada com o projeto OWASP Mobile Top 10. Neste projeto da OWASP pormenoriza cada categoria, ameaças, o vetor de ataque, falhas de segurança, impacto de negócio, como prevenir e alguns cenários. Um exemplo de uma possível categorização seria:

- Vulnerabilidade: *SQL Injection*.
- Ataque: Atacante aproveita de falhas em sistemas que interagem com bases de dados através da inserção de comandos *SQL*.
- Categoria: M1 – Um ataque de *SQL Injection* está inserido na categoria "Utilização impropria da plataforma".

O "**`owaspEngine`**" produz 3 relatórios referentes à análise a uma aplicação:

- O primeiro relatório produzido tem o nome de "`feedback`" e contém todas as vulnerabilidades detetadas pelas ferramentas de análise agrupadas por cada categoria do OWASP Mobile Top 10. E para cada vulnerabilidade apresentada, existe um conjunto de material educacional que permite ao programador perceber o impacto de cada vulnerabilidade e como corrigi-la. Este relatório é formatado em *JSON* e tem a seguinte estrutura:
  - "**`category`**": uma lista das categorias do OWASP Mobile Top 10 e para cada uma das categorias, um conjunto de vulnerabilidades que foram detetadas pelas ferramentas de análise para esta específica categoria.
  - "**`vulnerability`**": neste campo é apresentado o nome da vulnerabilidade.
  - "**`details`**": uma pequena descrição sobre a vulnerabilidade.
  - "**`severity`**": é identifica qual o nível de severidade da vulnerabilidade identificada.
  - "**`detectedby`**": uma lista das ferramentas que detetaram esta vulnerabilidade.
  - **"`url`"**: um conjunto de links de apoio educacional sobre a vulnerabilidade.

- "**vídeo**": um conjunto de vídeos de apoio educacional sobre a vulnerabilidade.
- "**book**": um conjunto de livros de apoio educacional sobre a vulnerabilidade.
- "**other**": outras informações relevantes podem ser adicionadas neste campo.

- O segundo relatório produzido tem o nome de "**feedback_levels**" e contém o número de vulnerabilidades por cada nível de severidade de uma aplicação móvel.
  Este relatório é também formatado em *JSON* e tem a seguinte estrutura:
  - "**value**": contém uma lista dividida por cada nível de severidade, contendo o número de vulnerabilidades identificadas com esse mesmo nível de severidade.
    - "**notice**": número de vulnerabilidades cujo nível de severidade é notice.
    - "**warning**": número de vulnerabilidades cujo nível de severidade é warning.
    - "**critical**": número de vulnerabilidades cujo nível de severidade é critical.

- O terceiro relatório produzido tem o nome de "**feedback_vulnerability_levels**" e contém uma lista de vulnerabilidades e o seu devido nível de severidade da uma aplicação analisada. Este relatório é formatado em *JSON* e tem a seguinte estrutura:
  - "**vulnerabilities**": uma lista de objetos *JSON* com os seguintes campos:
    - "**vulnerability**": é apresentado neste campo o nome da vulnerabilidade.
    - "**severity**": é apresentado neste campo o nível de severidade correspondente.

3.3.6. Uniformização dos dados

Quando uma ferramenta deteta uma vulnerabilidade é verificado no seu respetivo dicionário a que categoria pertence assim como o seu nível de severidade. No fim de cada análise, todas as ferramentas contêm um ficheiro com todas as vulnerabilidades que foram encontradas e categorizadas.

Após a categorização das vulnerabilidades, o sistema entra no processo de uniformização de dados. A uniformização de dados é feita através do mesmo *script* "**owaspEngine**". O que acontece é que cada ferramenta executada pelo sistema produz um relatório com todas as vulnerabilidades encontradas num ficheiro *JSON*. E quando o "**owaspEngine**" faz pré-processamento dos dados para produzir o relatório "`feedback`", são encontradas vulnerabilidades iguais, detetadas por ferramentas diferentes e com nomes diferentes.

E para que o relatório "`feedback`" não tenha informações repetidas, procede-se à uniformização dos dados. Para tal, o "**`owaspEngine`**" compara as vulnerabilidades encontradas por cada ferramenta e verifica no respetivo dicionário um campo denominado de "**`keywords`**", que contém palavras chave à cerca da vulnerabilidade em questão. Essas palavras chave do respetivo dicionário são comparadas com as palavras chave das vulnerabilidades encontradas por outras ferramentas nos respetivos dicionários, e se houver correspondência garante-se que as vulnerabilidades são iguais. Como se pode comprovar na Figura 9 que apresenta uma vulnerabilidade com a "**`keyword`**" xss, presente no dicionário do Androbugs, é a mesma vulnerabilidade da Figura 10 onde apresenta uma vulnerabilidade do dicionário do Super. Na prática, é eliminado todo o conteúdo do relatório "`feedback`" que seja igual (vulnerabilidades com as mesmas palavras chave) e no campo "**`detectedby`**" é adicionado o nome das ferramentas que encontraram aquela vulnerabilidade, como se pode verificar na Figura 11.

```
{
  "name": "WebView Potential XSS Attacks Checking",
  "category": "M1",
  "level": "Warning",
  "keywords": ["xss"]
},
```

Figura 9-Vulnerabilidade do dicionário Androbugs, com a keyword "xss"

```
{
    "name": "WebView XSS",
    "deatils": "Webview insecure implementation.
    "category": "M1",
    "keywords": ["xss"]
},
```

Figura 10-Vulnerabilidade do dicionário Super com a keyword "xss"

Figura 11-Normalização de dados

3.3.7 Base de Conhecimentos

A base de conhecimentos é uma base de dados dedicada a todo o conteúdo educacional para ajudar os programadores a saber mais sobre as vulnerabilidades, como corrigir e mitigar as mesmas. É uma base de dados que está em constante atualização, está num formato *JSON* e dividida pelas 10 categorias do OWASP Mobile Top 10.

A sua estrutura é definida da seguinte maneira:

- "**`category`**" – indica a categoria (M1 a M10)
- "**`links_by_category`**" – aqui é indicado uma lista com todos os links referentes à categoria em causa
- "**`books_by_category`**" – aqui é indicado uma lista com todos os livros referentes à categoria em causa
- "**`articles_by_category`**" – aqui é indicado uma lista com todos os artigos referentes à categoria em causa
- "**`keywords`**" – uma lista com todas as *keywords* extraídas dos dicionários pertencentes à respetiva categoria, e para cada *keyword* (que aponta para uma específica vulnerabilidade) um conjunto de informações adicionais
    - "**`name`**" – o nome da *keyword*
    - "**`links`**" – uma lista com todos os links sobre a vulnerabilidade
    - "**`books`**" – uma lista com todos os livros sobre a vulnerabilidade
    - "**`articles`**" – uma lista com todos artigos e jornais científicos sobre a vulnerabilidade

```
"category":"M1",
"links_by_category": ["https://www.owasp.org/index.php/Mobile_Top_10_2016-M1-Improper_Platform_Usage"],
"books_by_category": [],
"articles_by_cateory": [],
"keywords":[

    {
        "name": "contentProvider",
        "links": ["https://developer.android.com/guide/topics/manifest/manifest-intro.html","https://www.nowsecure
        "books": [],
        "articles": []
    },
    {
        "name": "androidManifest",
        "links": ["https://developer.android.com/reference/android/Manifest.permission.html","https://developer.an
        "books": [],
        "articles": []
    },
    {
        "name": "debuggable",
        "links": ["https://cwe.mitre.org/data/definitions/215.html","https://nvd.nist.gov/vuln/detail/CVE-2019-102
        "books": [],
        "articles": []
    },
    {
        "name": "fragment",
        "score": 4.6,
        "links": ["https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-6271","https://securityintelligence.com
        "books": ["https://securityintelligence.com/wp-content/uploads/2013/12/android-collapses-into-fragments.pd
        "articles": []
    },
```


Figura 12-Estrutura da base de conhecimentos

Para facilitar a consulta desta mesma base de conhecimentos aos programadores, foi desenvolvida uma interface gráfica com o objetivo de auxiliar e tornar o acesso à base de dados mais visual e facultar uma experiência ao programador mais amigável.

O desenvolvimento da arquitetura para esta interface gráfica tem duas componentes, a primeira componente é a parte visual, ou seja, o desenvolvimento do site e a segunda componente a API que gere os pedidos feitos pelo utilizador como se pode verificar na Figura 13 a arquitetura desta interface gráfica.

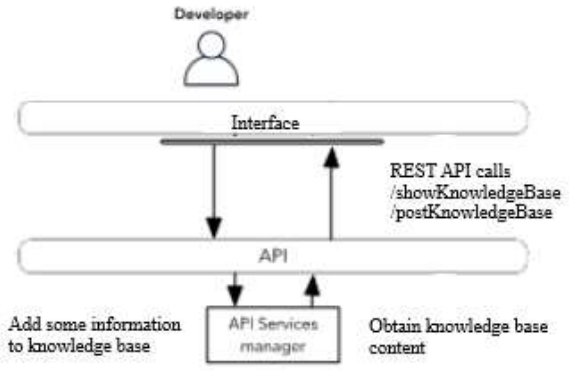

Figura 13-Arquitetura da interface gráfica

Ao abrir a interface gráfica na lateral esquerda pode-se verificar que existe a opção "`Knowledge Base`", ao carregar no botão, a página será encaminhada para uma outra na qual será apresentada toda a base de conhecimentos colecionada, ação feita através da chamada à API "**`showKnowledgeBase`**" que irá imprimir todos os conteúdos na página, todas as informações serão apresentadas numa tabela nas seguintes colunas:

- "**`category`**": a categoria na qual pertence a vulnerabilidade.
- "**`keywords`**": são apresentadas todas as *keywords* da vulnerabilidade correspondente
- "**`links`**": um grupo de links educacionais para perceber como funciona a vulnerabilidade, como corrigi-la ou mitigá-la.
- "**`books`**": uma lista de livros sobre a vulnerabilidade em questão é facultada para apoio aos programadores.
- "**`articles`**": um conjunto de artigos serão apresentados, na qual o assunto seja perceber o funcionamento da vulnerabilidade.

A página pode ser verificada na Figura 14.

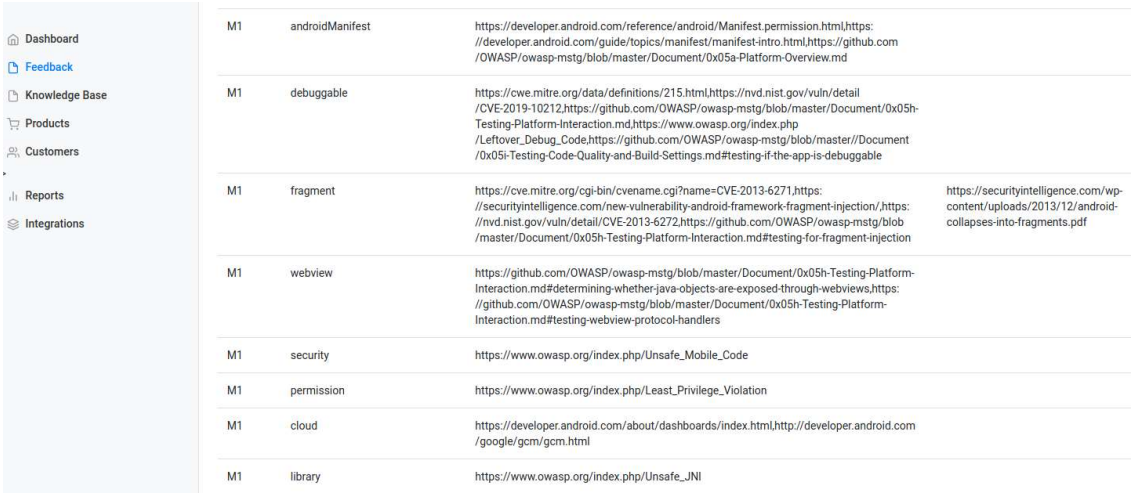

Figura 14-Página Knowledge Base da interface

Todas estas listas de informações adquiridas, sejam links, livros ou artigos, estão em constante crescimento. E o próprio utilizador pode ajudar a fazer crescer toda esta coletânea de informações que farão com que cada vez mais os programadores estejam conscientes dos perigos e da severidade de cada vulnerabilidade que este *software* deteta e também aprender as boas práticas de segurança e de desenvolvimento de programação. Para tal, esta interface gráfica permite o utilizador adicionar qualquer informação extra às *keywords* respetivas, preenchendo os seguintes campos, como se pode verificar na Figura 15:

1. Escolher em qual categoria do OWASP Mobile Top 10
2. Escolher um grupo de *keywords*. Estas *keywords* podem ser previamente consultadas na página "`Knowledge Base`".

3. Preencher este campo com um link que ache útil.
4. Preencher este campo com um nome de um livro que retrate as *keywords* ou a vulnerabilidade respetiva.
5. Por fim, preencher este campo com um artigo que ajude na perceção de como a vulnerabilidade funciona ou de como corrigi-la.

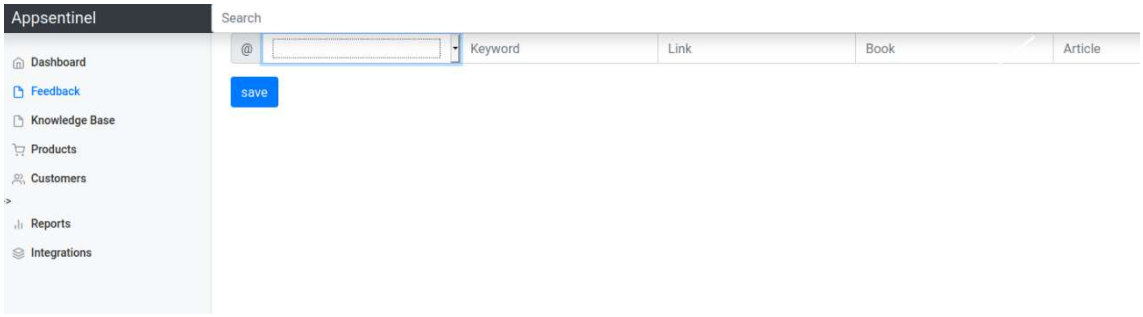

Figura 15-Adicionar informações na base de conhecimentos pela interface

Esta interface gráfica possui mais uma funcionalidade já desenvolvida, cujo nome da funcionalidade é "`Feedback`" que mostra uma tabela com o número de vulnerabilidades por categoria do OWASP Mobile Top 10 pertencentes a uma aplicação e também apresenta um gráfico circular com a percentagem de vulnerabilidades e com o nível de severidade "notice", "warning" e "critical" que uma aplicação contém.

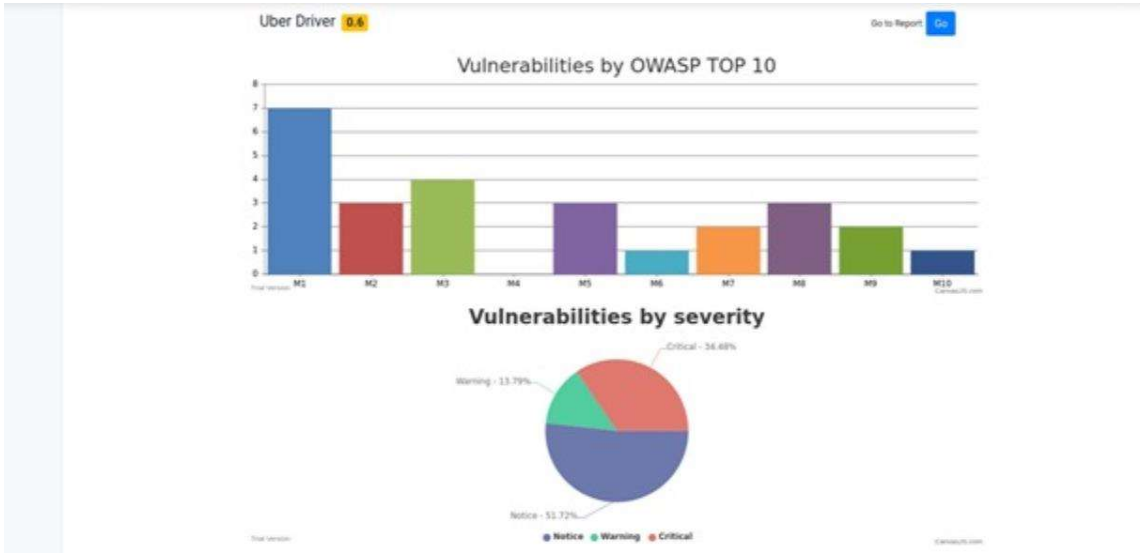

Figura 16-Página Feedback da interface

Algumas das funcionalidades desta interface poderão ser desenvolvidas num trabalho futuro, com o intuito de poder integrar outras caraterísticas que possam melhorar esta interface gráfica.

3.3.8 Relatório

O objetivo do sistema apresentado é automatizar o teste de APK e produzir um relatório detalhado para os programadores de aplicações móveis *Android* sobre vulnerabilidades existentes nas aplicações enviadas para análise. Sobre o relatório, o sistema fornece todas as informações resultantes das ferramentas de deteção de vulnerabilidades e que pode ajudar os programadores a aprender como mitigar essas vulnerabilidades. Também pode ajudar os gestores das *App Stores* a decidirem se aceitam ou rejeitam uma aplicação, tornando a própria loja mais segura para os utilizadores finais.

O sistema produz uma análise de vulnerabilidades de segurança, usando um conjunto de ferramentas de análise de vulnerabilidades de aplicações móveis. O sistema junta todas as informações resultantes de cada ferramenta e classifica e cada resultado segundo a metodologia OWASP Mobile Top 10 (os riscos variam de "M1" a "M10").

Esses resultados agregados são retornados ao programador (por meio da entrada da API REST "**`apkfeedback`**", conforme apresentado anteriormente) usando uma resposta formatada em *JSON* composta por várias estruturas diferentes.

Um ponto importante do relatório é o fato de ser educacional para programadores, pois para cada vulnerabilidade encontrada, os programadores recebem um conjunto de informações necessárias para mitigar as vulnerabilidades ou corrigi-las (*links*, vídeos, livros e outros recursos) e que lhes permitirá entender melhor as vulnerabilidades encontradas, como funcionam e como os atacantes as exploram.

A resposta que contém os resultados da análise feita e o relatório educacional é uma mensagem formatada em *JSON* composta por várias estruturas - "**`status`**", "**`results_history`**" e "**`results`**". O primeiro, "**`status`**", é um objeto *JSON* que contém um valor booleano que indica se a solicitação da API foi bem-sucedida ou não.

O segundo elemento no resultado é o "**`results_history`** " composto por uma matriz de objetos *JSON* que representam o histórico das diferentes análises de vulnerabilidades de uma determinada aplicação móvel. O histórico de resultados é fundamental porque permite acompanhar o nível de segurança de uma determinada aplicação ao longo do tempo e verificar se essa mesma aplicação é melhorada quanto à sua segurança. Cada um dos elementos do "**`results_history`**" tem a seguinte estrutura:

- "**`created at`**": representa a hora e a data da realização da análise;

- "**details**": uma sequência de dados que fornece detalhes extras sobre a análise realizada. Estes detalhes podem incluir informações sobre algumas especificidades das ferramentas usadas;
- "**id**": cada análise de vulnerabilidades é representada e identificada de maneira única pelo sistema.
- "**apkid**": um identificador único do APK - depende da *App Store* (md5);
- "**results location**": o servidor armazena os diferentes resultados produzidos pelas múltiplas ferramentas de busca de vulnerabilidades utilizadas.
- "**scantools**": representa a identificação das múltiplas ferramentas de deteção de vulnerabilidades específicas usadas para realizar a análise de segurança e identificação de vulnerabilidades;
- "**status**": um valor booleano que indica o sucesso da operação;

Após concluídas todas as operações acima indicadas, o sistema irá executar a última operação, ou seja, produzir um relatório detalhado para os programadores de aplicações móveis *Android* sobre vulnerabilidades existentes nas suas aplicações e aprender como mitigar essas vulnerabilidades, isto com o objetivo de reduzir o número falhas de segurança em aplicações *Android*.

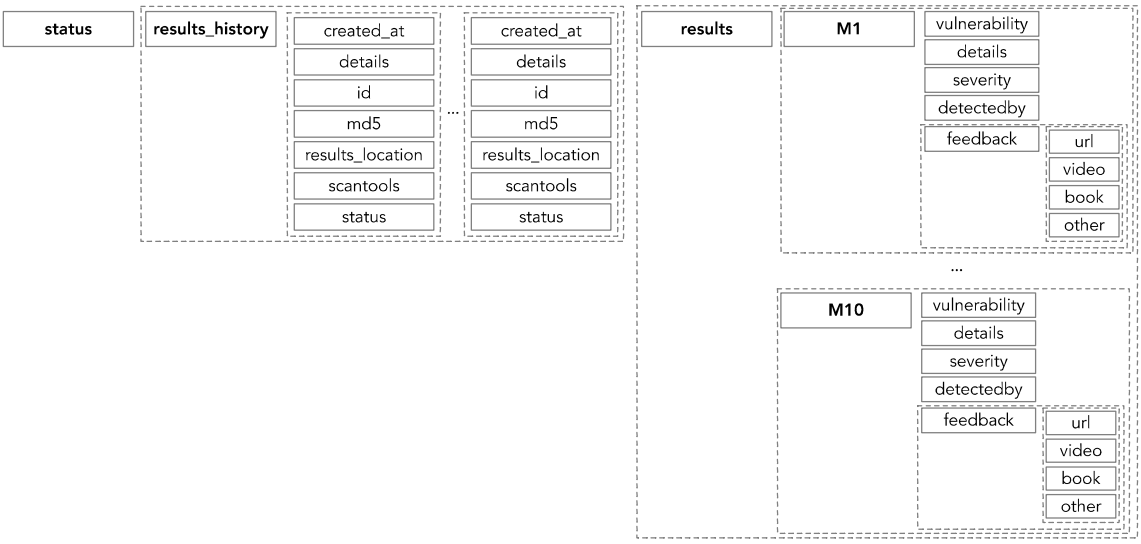

Figura 17-Estrutura de mensagem formatada em JSON do relatório (Fonte projeto AppSentinel)

Para tal, o sistema fornece um relatório educacional útil aos programadores, com informação detalhada para cada uma das vulnerabilidades identificadas. O relatório fornece informação adicional aos programadores (*links*, vídeos, livros e outros recursos, provenientes da base de conhecimentos descrita anteriormente) que lhes permitirá entender melhor as vulnerabilidades que foram encontradas, como as mesmas funcionam e como são exploradas pelos atacantes, e como corrigi-las ou mitigá-las. O relatório que é entregue ao programador contém todas as vulnerabilidades categorizadas segundo o OWASP Mobile Top 10 e seguidamente normalizadas

e para cada vulnerabilidade um conjunto de informações que irá ajudá-lo a mitigar a vulnerabilidade em questão.

O último elemento da resposta é o "**`results`**" que vem com todas as informações encontradas pelas ferramentas:

- A categoria - indica a que categoria (segundo o OWASP Mobile Top 10) a vulnerabilidade se encontra (pode ir de M1 a M10). Dentro da categoria uma lista com todas as vulnerabilidades, sendo a primeira informação
- "**`vulnerability`**" - indica o nome da vulnerabilidade.
- "**`details`**" – aqui são apresentadas informações detalhadas sobre a vulnerabilidade.
- "**`severity`**" - indica o nível de severidade que a vulnerabilidade pode ter. As vulnerabilidades de nível mais baixo apresentam o nível "notice", o segundo nível "warning" e por último, o nível mais elevado "critical". Estes níveis permitem que os programadores entendam rapidamente os riscos de segurança das suas aplicações *Android* e aprendam a tomar as devidas ações para mitigá-los. Quanto maior é o nível de severidade maior é a probabilidade de um atacante poder explorar essa vulnerabilidade e conseguir obter várias informações sobre o utilizador.
- "**`detectedby`**" – indica quais as ferramentas que detetaram aquele problema de segurança.
- "**`feedback`**" – um dos principais objetivos deste sistema é fornecer um relatório apropriado aos programadores, fornecendo informações não apenas sobre as vulnerabilidades de segurança existentes nas aplicações *Android*, mas também informações sobre o modo como as vulnerabilidades detetadas operam, como elas afetam a segurança dos utilizadores finais e também a possibilidade de corrigi-las ou mitigá-las. Portanto, esse mecanismo visa educar não apenas sobre riscos de desenvolvimento, mas também sobre boas práticas de desenvolvimento que precisam ser definidas para corrigir esses riscos de segurança de desenvolvimento. Aqui serão apresentados todos os conteúdos reunidos para mostrar ao programador como resolver e aprender sobre a vulnerabilidade:
  - "**`url`**" – *links* que levam a vários recursos online sobre a vulnerabilidade.
  - "**`vídeo`**" – *links* que direcionam para alguns vídeos que contém informação relevante sobre a vulnerabilidade.
  - "**`book`**" – livros sobre a vulnerabilidade.
  - "**`other`**" – qualquer outro conteúdo à cerca da vulnerabilidade, como artigos científicos, jornais, etc…

O relatório é o resultado de uma chamada a API para obter todas as informações sobre a análise à devida aplicação *Android*.

3.4 Tecnologias utilizadas

Neste capítulo serão exploradas e identificadas as tecnologias com que este projeto se concretizou.

Para este projeto a linguagem de programação mais utilizada foi *Python* (ambas as versões 2 e 3). *Python* é uma linguagem de programação criada por Guido van Rossum em 1991. É uma linguagem de alto nível, fácil legibilidade de código, grande velocidade de execução, interpretada, de *script* e orientada a objetos.

Quase todo o desenvolvimento do projeto é feito em *Python*, por exemplo a API, o processamento das aplicações e quase todas as ferramentas. As ferramentas de análise de vulnerabilidades que foram desenvolvidas em *Python*, uma foi desenvolvida com a versão 2 (a ferramenta Androbugs) e a outra com a versão 3 (a ferramenta DroidStatx). Por este motivo é que se utilizou as duas versões do *Python*.

Outra tecnologia utilizada para o desenvolvimento do sistema foi o *Rust*, nomeadamente a ferramenta de análise de vulnerabilidades Super Android Analyzer. *Rust* é uma linguagem de programação compilada, funcional e orientada a objetos. Começou a ser desenvolvida por Graydon Hoare, funcionário da Mozilla, até que em 2009 a organização para qual Graydon trabalhava começou a apoiar o seu projeto.

Para a construção da base de dados utilizou-se a tecnologia *MYSQL*. Desenvolvida pela Oracle Corporation e lançada em 1995, é uma tecnologia que gere base de dados e utiliza a linguagem de programação *SQL* (uma linguagem de consulta estruturada).

No desenvolvimento da interface gráfica foi utilizada a tecnologia *AngularJS*. É uma *framework* para desenvolvimento de aplicações web dinâmicos. O *AngularJS* permite que se utilize o *HTML* para desenvolver a estrutura visual da aplicação. É também uma tecnologia que é baseada em componentes.

Por último, utilizou-se a tecnologia *Docker*. Uma plataforma *open source* de desenvolvimento e execução de aplicações e tem como base a linguagem de programação Go da Google. A sua principal função é a facilidade de criar e gerir ambientes isolados com recurso à tecnologia de "`containers`", ou seja, isolar uma aplicação num ambiente dentro desse "`container`", e que pode ser executado em qualquer outra máquina desde que tenha o *Docker* instalado.

3.5 Conclusões

O sistema desenvolvido analisa um conjunto de aplicações que são previamente inseridas na base de dados através da API. No fim de correr da análise são identificados todos os problemas e vulnerabilidades, de seguida é entregue ao programador um relatório detalhado com todas as

informações à cerca das vulnerabilidades detetadas. Este relatório vai ajudar o programador a desenvolver aplicações seguras e consequentemente o utilizador final terá mais segurança na utilização da mesma.

Este sistema automatizado, mas possui algumas limitações, por exemplo, o programador terá sempre de confirmar manualmente se existe mesmo as vulnerabilidades que foram detetadas, ou seja, pode haver falsos positivos. Por isso, este sistema deteta problemas e vulnerabilidades da aplicação que têm de ser sempre confirmados pelo programador.

CAPÍTULO 4

Testes e discussão de resultados

Esta dissertação tem como objetivo apresentar um sistema capaz de analisar aplicações *Android* para dispositivos móveis e por fim retornar ao programador um relatório com todas as falhas de segurança e vulnerabilidades encontradas, categorizadas e normalizadas segundo o OWASP Mobile Top 10. Para cada vulnerabilidade encontrada pelo sistema, um conjunto de informações sobre a mesma com a finalidade de corrigir ou mitigar essa mesma vulnerabilidade, para ajudar os programadores a desenvolverem aplicações móveis em segurança.

Neste capítulo pretende-se introduzir um conjunto de dados utilizados e obtidos pelo sistema na análise das vulnerabilidades, no comportamento das ferramentas, na categorização das vulnerabilidades e no comportamento do sistema em identificar várias ferramentas para a mesma vulnerabilidade e por fim comparar e comentar os resultados obtidos.

Os testes foram executados num único servidor equipado com AMD Ryzen 7 Pro 3700 correndo a 4.4GHz com 64GB de DDR4 ECC 2666MHz RAM. O servidor também estava equipado com 2x SSD NVMe 960GB e estava a correr num Ubuntu Server 18.04 com 4.15.0-109-generic kernel como sistema operativo.

Foram realizados vários testes de modo a garantir a fiabilidade do sistema. De modo a poder comparar o nível de segurança entre *App Stores*, foram realizados testes a três diferentes *App Stores*: Aptoide, Play Store e APKPure. Estas *App Stores* foram escolhidas por serem as principais *App Stores* e as mais utilizadas.

Primeiro foram selecionadas as aplicações com mais *downloads* por categoria de cada *App Store* (42 categorias no Aptoide, 33 categorias na Google Play Store e 32 categorias no APKPure). Depois de processadas todas as aplicações móveis, obteve-se um total de 1065 aplicações.

Cada aplicação tinha em média 27MB de tamanho, sendo que a aplicação *Android* com mais memória ocupada foi de 1,37GB e a menor foi de 27KB.

O sistema estava configurado para usar três diferentes ferramentas de análises de vulnerabilidades - Androbugs, Droidstatx e Super Android Analyzer. Foram escolhidas estas ferramentas, por serem ferramentas que executam uma análise estática e que geram resultados adequados para o que se pretende nesta dissertação.

4.1 Recolha de dados

Para a obtenção dos resultados, foram analisadas cerca de 1065 aplicações *Android* de diferentes *App Stores* (Aptoide, Play Store, APKPure), ou seja, as aplicações *Android* com mais *downloads* de todas as categorias que existentes das diferentes *App Stores*. Os testes foram realizados no dia 04/08/2020.

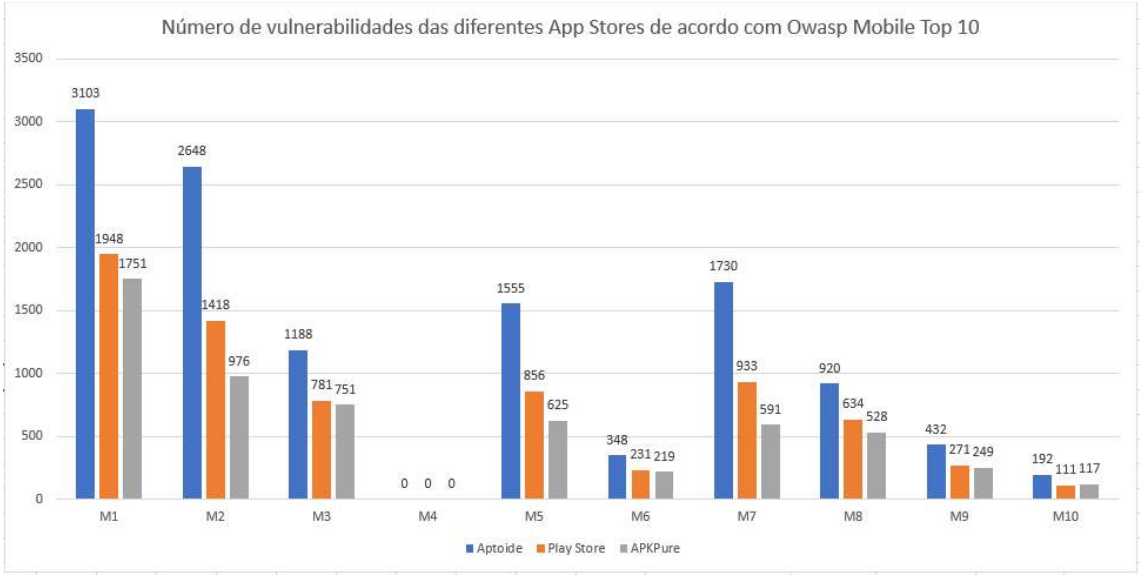

Gráfico 1-Número de vulnerabilidades das diferentes App Stores de acordo com Owasp Mobile Top 10

O Gráfico 1 apresenta o número de vulnerabilidades encontradas das 1065 aplicações *Android* das diferentes *App Stores* por categoria do OWASP Mobile Top 10.

Através da análise ao Gráfico 1, verifica-se que as categorias propícias a apresentarem mais vulnerabilidades são (M1) Utilização impropria da plataforma, (M2) Armazenamento Inseguro de Dados, (M5) Criptografia Insuficiente e (M7) Má Qualidade de Código.

Houve uma categoria que não foram detetadas nenhumas vulnerabilidades, que foi o caso da (M4) Autenticação Insegura, o que pode indicar que os programadores têm uma certa preocupação para questões deste âmbito.

De todas as 1065 aplicações *Android* das diferentes *App Stores* analisadas, foram identificadas 25106 vulnerabilidades, das quais 8356 são vulnerabilidades com nível de severidade notice, 11087 são vulnerabilidades com nível de severidade warning e por último 5663 vulnerabilidades com nível de severidade critical, como mostra o Gráfico 2.

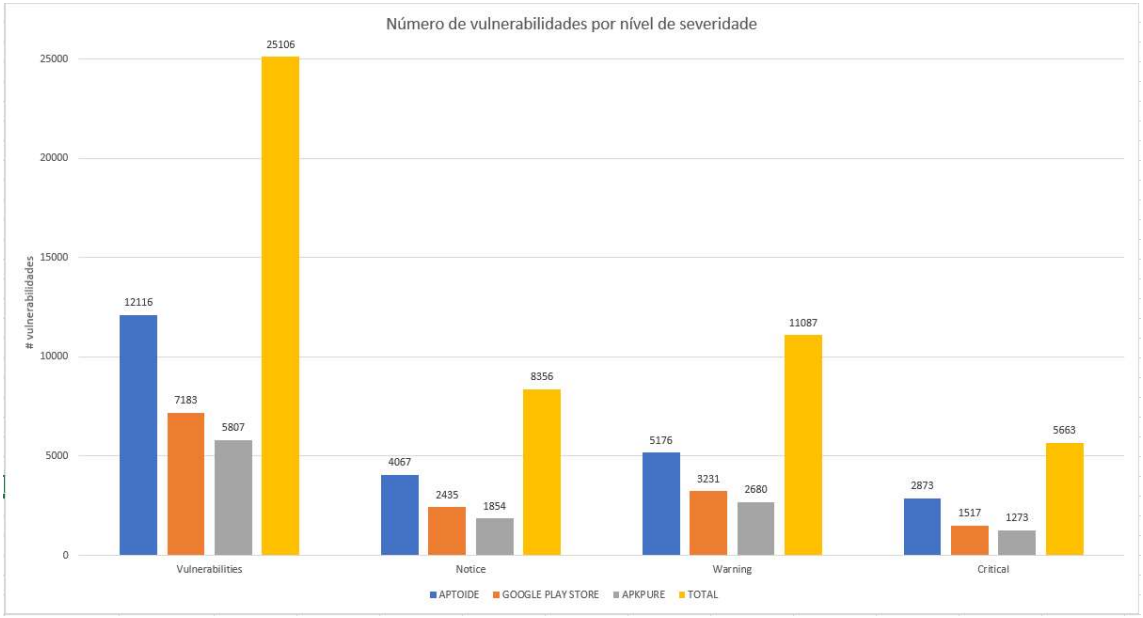

Gráfico 2-Número de vulnerabilidades por nível de severidade

Em termos de categorias com mais vulnerabilidades detetadas, é difícil de fazer uma comparação visto as categorias de cada *App Store* são totalmente diferentes. No caso da Aptoide as categorias com mais vulnerabilidades identificadas são "BEAUTY", "WEATHER", "ENTERTAINMENT" e "EDUCATION", no caso da Play Store são "COMMUNICATION", "GAME", "FAMILY" e "ART AND DESIGN", e por fim a APKPure são "COMMUNICATION", "MAPS AND NAVIGATION", "PRODUCTIVITY" e "TOOLS".

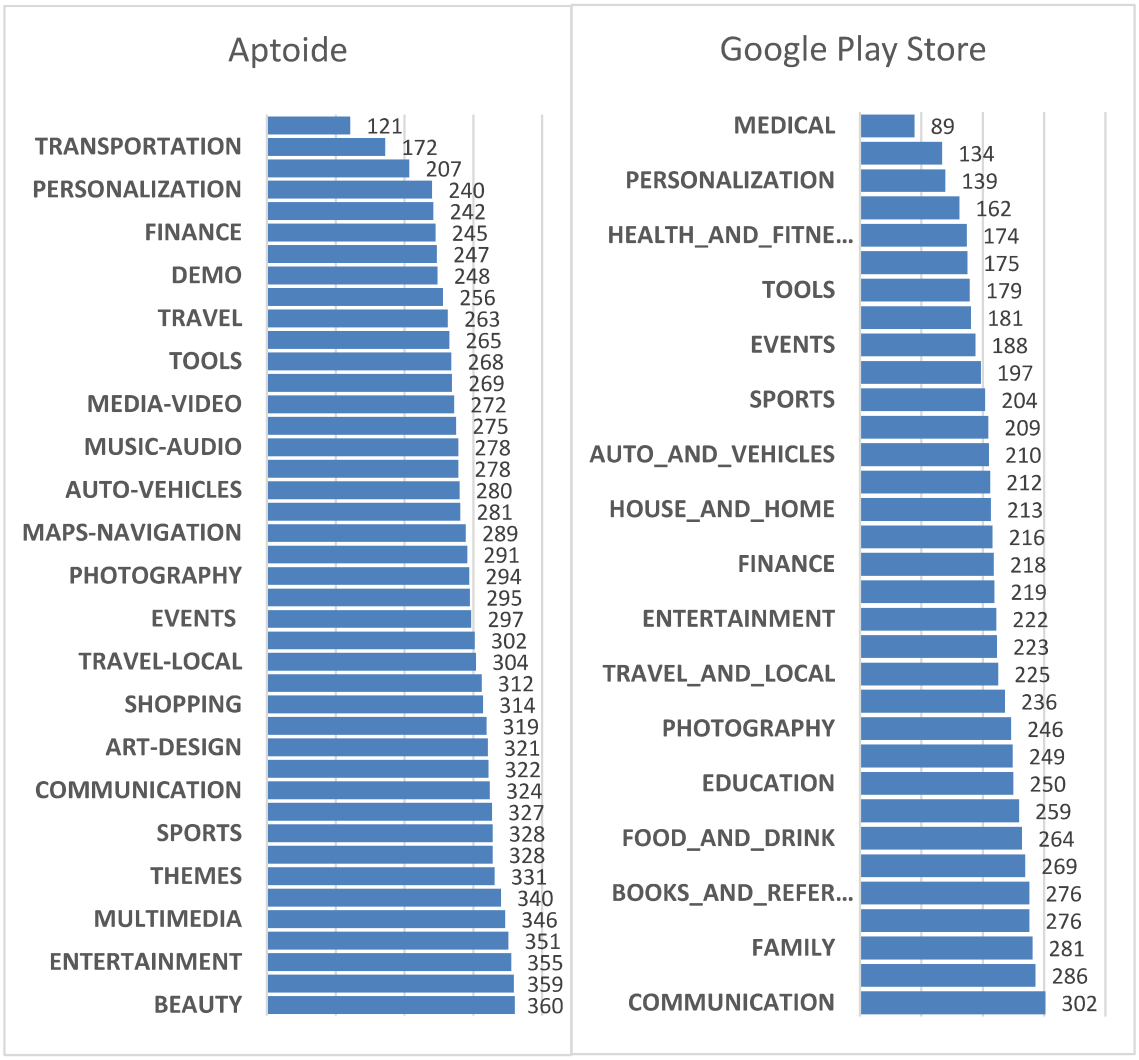

Gráfico 3-Número de vulnerabilidades por categoria (Aptoide) Gráfico 4-Número de vulnerabilidades por categoria (Play Store)

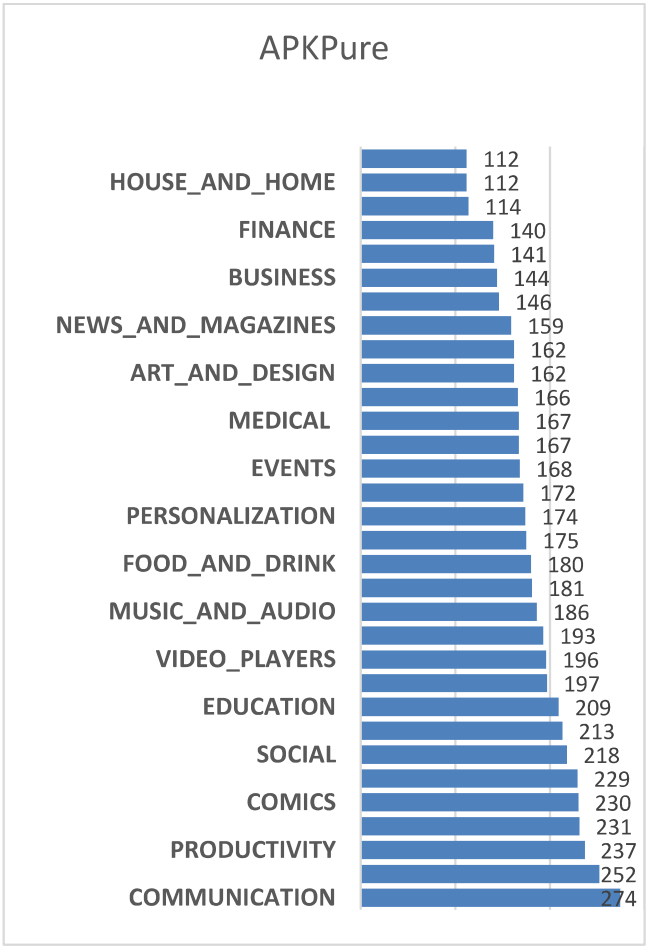

Gráfico 5-Número de vulnerabilidades por categoria (APKPure)

Ao comparar as categorias com mais vulnerabilidades detetadas de cada *App Store*, chega-se à conclusão que existe uma parecença entre elas, como se pode comprovar na Tabela 4. No primeiro grupo foi detetado parecenças entre a categoria "BEAUTY" da Aptoide com a categoria "FAMILY" da Play Store. No segundo grupo entre a categoria "ENTERTAINMENT" da Aptoide com a categoria "GAME da Play Store. Por fim, o terceiro grupo entre as categorias "COMMUNICATION" das *App Stores* da Play Store e APKPure.

GRUPO	CATEGORIA 1	CATEGORIA 2
Grupo 1	BEAUTY	FAMILY
Grupo 2	ENTERTAINMENT	GAME
Grupo 3	COMMUNICATION	COMMUNICATION

Tabela 4-Categorias semelhantes

Para realizar estes testes massivos e analisar todas as aplicações *Android* foram precisas 7 horas e 42 minutos. O tempo médio de cada análise foi de 29 segundos. O tempo máximo de análise registado foi da Aptoide com 4 minutos e 24 segundos, enquanto o tempo mínimo de

análise foi tanto da Aptoide como da APKPure com 1 segundo. Na Tabela 5 mostra os tempos máximos, mínimos e médios que uma aplicação levou para ser analisada.

	Aptoide	Google Play Store	APKPure	Global
Total	03:11:31	02:28:53	02:07:08	07:47:32
Max	00:04:24	00:04:10	00:03:56	00:04:10
Min	00:00:01	00:00:02	00:00:01	00:00:01
Average	00:00:28	00:00:36	00:00:24	00:00:29

Tabela 5-Tempos de análise das diferentes App Stores

Androbugs	Super Android Analyzer	DroidStatx
00:00:16	00:00:02	00:00:14

Tabela 6-Tempo médio em que cada ferramenta analisa cada aplicação móvel (segundos)

A Tabela 6 apresenta o tempo médio que cada *plugin* de análise de vulnerabilidades no sistema levou a testar uma aplicação *Android*. Para este teste, o tempo médio que ferramenta Androbugs levou para correr uma aplicação foi de 16 segundos, o tempo médio que a ferramenta Super Android Analyzer levou para correr uma aplicação foi 00:00:02s e por último o tempo médio que a ferramenta DroidStatx levou para correr uma aplicação foi 00:00:14s.

O Gráfico 6 mostra o número de vulnerabilidades encontradas por conjunto de ferramentas de análise de deteção de vulnerabilidades, ou seja, uma certa vulnerabilidade que foi detetada apenas por uma ferramenta, por duas ferramentas e por três ferramentas. Neste caso existiram cerca de 18515 vulnerabilidades que foram encontradas por apenas 1 ferramenta de deteção, 4129 vulnerabilidades que foram encontradas por duas ferramentas e 459 que foram encontradas por apenas três ferramentas. Isto mostra que nem todas as ferramentas detetam as mesmas vulnerabilidades.

Gráfico 6-Número de vulnerabilidades encontradas por conjunto de ferramentas

4.2 Comparação dos resultados

Após a análise 1056 aplicações *Android*, foram identificas ao todo 25106 vulnerabilidades detetadas nas três *App Stores* diferentes. Das 25106 vulnerabilidades identificadas 8356 são vulnerabilidades com nível de severidade notice, 11087 são vulnerabilidades com nível de severidade warning e por último, 5663 vulnerabilidades com nível de severidade critical, como mostra o Gráfico 2. O que significa que em média por cada aplicação existe 24 vulnerabilidades, das quais oito são vulnerabilidades de nível de severidade notice, dez são vulnerabilidade do nível warning e seis vulnerabilidades do nível de severidade critical.

Isto mostra que os programadores ainda não têm consciência dos riscos que as suas aplicações móveis enfrentam. Existe um grande número de vulnerabilidades principalmente críticas que resultam em grandes falhas de segurança, fazendo com um atacante tente explorar e consiga extrair informações pessoais com maior facilidade.

Ainda na continuação da análise às aplicações *Android*, uma das características mais importantes é verificar se o sistema é eficaz quanto à velocidade na execução de todas as ferramentas, como se pode verificar na Tabela 5, no fim de concluir a análise de todas 1056 aplicações *Android* verificou-se que o tempo total para executar todas as ferramentas propostas foi de 07:47:32s, fazendo com que o tempo médio de análise por aplicação *Android* foi 00:29s. A Tabela 5 mostra que existe uma grande disparidade entre tempos máximos e mínimos, mas estes são determinados pelo tamanho da aplicação *Android* que é analisada.

A Tabela 6 apresenta a ferramenta Super como a mais rápida no processo de análise com a média de 2 segundos por aplicação móvel, enquanto as ferramentas Androbugs e Droidstatx

foram um pouco mais lentas, com 16 segundos e 14 segundos respetivamente. Isto não quer dizer que a ferramenta Super seja melhor, apenas as ferramentas Androbugs e DroidStatx são mais complexas e apresentam relatórios mais extensivos.

A velocidade foi um critério também importante na escolha de ferramentas para o sistema, pois era preciso ferramentas com alta fiabilidade, no sentido em que geravam resultados satisfatórios, por outro lado, o tempo de execução seja diminuto, pois não é esperado que com a acumulação de aplicações *Android* que o sistema tem para analisar, o tempo total de execução tender para grandes valores, não se iria tornar prático.

Passando para a parte da categorização, verificou se que existe mais vulnerabilidades nas categorias M1, M2, M3, M5 e M7 do OWASP Mobile Top 10, como se pode concluir no Gráfico 1.

Grande parte das aplicações *Android* possuem vulnerabilidades na categoria (M1) Utilização impropria da plataforma, com 6082 vulnerabilidades encontradas. Vulnerabilidades inseridas nesta categoria têm falhas de segurança devido ao uso indevido do sistema operativo ou na falta de conhecimento sobre os controles de segurança da plataforma. Isso pode incluir "*Android intents*" - é basicamente uma mensagem que é passada entre componentes (por exemplo *activities* e *services*), *keychain* ou outros controles de segurança que fazem parte da plataforma. A ocorrência de vulnerabilidades categorizadas em M1 é bastante comum, pois não houve nenhuma aplicação que tivesse pelo menos uma vulnerabilidade e que não fosse categorizada nesta mesma categoria.

A segunda categoria com mais vulnerabilidades detetadas foi a categoria (M2) Armazenamento Inseguro de Dados, com 5042 vulnerabilidades. Este tipo de vulnerabilidade faz com que a exploração de dados não seguros seja possível devido à falta de conhecimento dos programadores de como um dispositivo móvel armazena dados em cache, imagens e buffers. Um atacante examina vários locais num dispositivo que tenha estas vulnerabilidades, em busca de dados não protegidos. Estes dados podem ser bases de dados SQL, ficheiros de log, dados XML, dados binários, cookies, cartões SD e dados na *cloud*.

A próxima categoria a relatar é a (M7) Má Qualidade de Código, com 3254 vulnerabilidades, pode-se verificar que um grande número de vulnerabilidades encontradas nesta categoria reflete-se na falta de boas práticas de programação, por exemplo, a falta de documentação para numa equipa que desenvolve uma aplicação, pode fazer com que haja inconsistências no código final. É possível que um atacante com algumas ferramentas automáticas, consiga identificar alguns problemas na memória, *buffer overflows* e outros… e com isto consiga obter algumas informações privilegiadas.

De seguida a categoria na qual se verificou um grande número de vulnerabilidades inseridas, foi a categoria (M5) Criptografia Insuficiente, com 3036 vulnerabilidades encontradas, o estudo indica que os dados nas aplicações móveis se tornam mais vulneráveis devido aos fracos algoritmos de criptografia ou a falta deles. Atacantes podem obter acesso ao dispositivo móvel se utilizarem ferramentas para espiar o tráfego de rede para aceder a informações encriptadas.

Por fim, a quinta categoria que se destaca com mais vulnerabilidades é a categoria (M3) Comunicação Insegura, com 2720 vulnerabilidades detetadas. As vulnerabilidades desta categoria costumam ter falhas na transmissão de dados, tanto a enviar como a receber dados para a aplicação móvel. Um atacante pode intercetar os dados através de um *malware*, isto se a aplicação estiver comprometida.

Por último, foi testado o comportamento das "*keywords*", ou seja, quantas e quais ferramentas detetaram uma específica vulnerabilidade. Como o trabalho de implementar as "*keywords*" nos dicionários das respetivas ferramentas é estritamente manual, através do estudo antecipado de cada vulnerabilidade e das "*keywords*" de todos os dicionários, pode-se concluir que a percentagem de acerto de duas ou mais ferramentas detetarem uma vulnerabilidade com a "*keyword*", por exemplo, "*adb*" e essas vulnerabilidades serem consideradas iguais é de 100%. O número de vulnerabilidades detetadas por apenas uma ferramenta foi de 18515 vulnerabilidades, por duas ferramentas foi 4129 vulnerabilidades e por três ferramentas foi de 459 vulnerabilidades.

CAPÍTULO 5

Conclusões

Esta secção tem o objetivo de retirar as principais conclusões obtidas com esta dissertação, no desenvolvimento do sistema proposto, nos resultados gerados e as suas limitações. Também introduzir possíveis trabalhos académicos para investigação que possam ser desenvolvidos no futuro, assim como falar nas implicações desta dissertação ao nível empresarial.

5.1. Conclusões

Todos os dias existe um número elevado de aplicações a serem desenvolvidas, inseridas em diferentes categorias, do *fitness* aos jogos, mas com tantas aplicações a serem colocadas nas *App Stores* haverá cada vez mais e variadíssimas falhas de segurança.

A despreocupação do programador em verificar se a aplicação pode ficar comprometida, o pouco tempo que pode existir para o desenvolvimento da aplicação móvel ou até mesmo a falta de conhecimentos de boas práticas de programação, fazem com que o utilizador final fique mais vulnerável a possíveis ataques. Para evitar este cenário, o sistema proposto, que tem o objetivo de juntar as melhores ferramentas de análise de vulnerabilidades *open source*, categorizar as mesmas, normalizar os dados e entregar ao programador um relatório final com todas as vulnerabilidades categorizadas segundo o OWASP Mobile Top 10 e para cada vulnerabilidade apresentar um conjunto de informações educacionais, que permitem corrigir a falha ou mitigá-la teve sucesso. Portanto, à questão de investigação proposta na devida secção, "Será possível desenvolver um sistema que ajude os programadores a identificar e corrigir os problemas de segurança das aplicações móveis para aumentar a segurança das mesmas?", a resposta é sim. E como este sistema foi desenvolvido para ser um sistema de plugins, facilmente se tira uma ferramenta e se coloca outra, isto torna o trabalho final muito mais robusto.

Os resultados obtidos através de testes massivos a várias aplicações móveis mostram que existe uma grande despreocupação por parte dos programadores no desenvolvimento de aplicações móveis.

Existe até mesmo uma certa falha por parte das *App Stores* por não terem um certo cuidado no momento em que essas aplicações com grandes problemas de segurança ficam disponíveis para qualquer utilizador as descarregar, levando a que muitos utilizadores sem conhecimento lhes seja roubada informações confidenciais.

Os valores adquiridos pelos testes feitos ao sistema exibem uma pequena realidade do que se passa no mundo dos dispositivos móveis, em que geralmente um utilizador com *smartphone* possui muitas aplicações no seu aparelho.

Com este trabalho espera-se que exista uma diminuição destes valores tão negativos referentes às vulnerabilidades encontradas nas aplicações móveis, dar a conhecer ao programador o conhecimento necessário para poder desenvolver o seu trabalho com segurança através de todo o apoio educacional que o sistema faculta e que pode estar a ser constantemente atualizado e por fim, e possivelmente o ponto mais importante, dar ao utilizador final a confiança em poder descarregar uma aplicação sem que lhe seja roubada qualquer informação e estar completamente protegido.

A nível empresarial este projeto tem muito a oferecer. Qualquer *App Store* ou empresas de desenvolvimento de aplicações móveis podiam ser boas opções para este sistema funcionar e ter sucesso. Poderia ser aplicado como um serviço de *pentesting* e identificação automática de vulnerabilidades de segurança a aplicações móveis *Android*, por exemplo. Tudo o que possa haver com segurança nas aplicações *Android* e potenciar conhecimentos dos respetivos programadores, em termos de negócio haveria mercado. Para este nível algumas mudanças teriam de acontecer, por exemplo o facto de estar constantemente a atualizar as ferramentas, pois novas vulnerabilidades vão aparecendo todos dias. Não seria coerente nem responsável ter um sistema com ferramentas de análise desatualizadas que não seriam capazes de analisar aplicações mais recentes, por outro lado, ter de se criar manualmente os dicionários das ferramentas faz com que seja um pouco exaustivo a adição de novas ferramentas ao sistema, uma opção era tornar o sistema um pouco mais automático. Com estes pontos de atualização e progressão do sistema, seria uma opção bastante válida para o mercado da segurança das aplicações *Android*.

No desenvolvimento deste projeto foram escritos dois artigos, o primeiro artigo submetido (aprovado), que tem como base esta dissertação, cujo título é "*Automated security testing of Android applications for secure mobile development*", fala sobre o desenvolvimento de um sistema, que foi desenhado para ajudar as *App Stores Android* (que para este caso é a Aptoide) a detetar vulnerabilidades em aplicações enviadas para a sua *App Store*, e por fim fornecer um relatório detalhado para os programadores, tornando-os mais cientes dos processos de desenvolvimento seguros, melhorando a qualidade e segurança das suas aplicações, antes de serem disponibilizados aos utilizadores finais e instalados nos seus dispositivos. O segundo artigo submetido (ainda em avaliação) com o título "*Improving Android applications quality through scalable automated security testing*" e tem como finalidade especificar e testar um sistema que foi desenvolvido para melhorar a segurança das aplicações *Android*. Este sistema é baseado no teste automatizado de aplicações enviados para a *App Store* e na identificação de potenciais vulnerabilidades de segurança como uma forma de melhorar o processo de desenvolvimento de aplicações, resultando na melhoria global da segurança das aplicações tanto na *App Store* como nos dispositivos dos utilizadores finais.

Este sistema pode ser melhorado por quem quiser continuar este projeto, sendo desenvolvido em *open source*, onde o pode encontrar no repositório do github (https://github.com/franciscopalma/appsentinel).

5.2. Limitações do sistema

Como já referido na secção anterior uma das limitações do sistema é referente aos dicionários das ferramentas de análise de vulnerabilidades. Por cada vez que se adiciona uma ferramenta para identificação de vulnerabilidades, é necessário com ela criar o plugin respetivo e o dicionário referente a essa mesma ferramenta. Criar um dicionário torna se demoroso e exaustivo. Um dicionário é um ficheiro com formatação *JSON* em que todos os campos necessitam ser preenchidos, serem colocados todas as vulnerabilidades que a ferramenta possa identificar, acrescentar o nível de severidade de cada vulnerabilidade e por fim categorizar cada uma delas segundo o OWASP Mobile Top 10, isto tudo manualmente. Este processo torna o sistema um pouco difícil para quem pretende adicionar mais ferramentas e não quer se preocupar com o facto de ter de categorizar e adicionar vulnerabilidades a um ficheiro.

5.3. Propostas de investigação futuras

O trabalho de investigação mais interessante que pode ser proposto futuramente, tem a ver a evolução e a melhoria do sistema desenvolvido para esta dissertação. Para começar, a resolução das limitações do sistema e dos problemas daí provenientes, poderia ser bastante apelativo, com a adição de várias técnicas de *machine learning* e *text mining* para a automatização da categorização das vulnerabilidades identificadas pelas ferramentas de análise, com isto deixava o sistema completamente livre da criação manual dos dicionários.

Outra proposta de investigação seria o desenvolvimento de técnicas de computação distribuída para a melhoria da velocidade de todo o sistema da AppSentinel, fazendo com que o download das aplicações *Android* e a análise de todas as ferramentas de busca de vulnerabilidades tivessem um desempenho muito melhor, tornando o sistema mais robusto a ponto de poder receber os milhares de aplicações que são sugeridas às *App Stores* provenientes dos programadores.

Referências Bibliográficas

C, André. (2019). *DroidStatx* (p. https://github.com/clviper/droidstatx). p. https://github.com/clviper/droidstatx.

Becher, M., Freiling, F. C., Hoffmann, J., Holz, T., Uellenbeck, S., & Wolf, C. (2011). Mobile security catching up? Revealing the nuts and bolts of the security of mobile devices. *Proceedings - IEEE Symposium on Security and Privacy*, (November), 96–111. https://doi.org/10.1109/SP.2011.29

Brahler. S, (2010), "Analysis of the Android Architecture", *Os.Ibds.Kit.Edu*, p. 52.

Braga, Alexandre. (2012). Introdução à Segurança de Dispositivos Móveis Modernos–Um Estudo de Caso em Android. Simpósio em Segurança ldots.

Cochereau, A., (2008). [First look and early attachment of the newborn]. *Soins. Pediatrie, puericulture*, (257), p.8.

Debize, T. (2019). *Androwarn* (p. https://github.com/maaaaz/androwarn). p. https://github.com/maaaaz/androwarn.

Dumais, S. (2005). Latent Semantic Analysis. *Annual Review of Information Science and Technology*, *38*, 188–230.

Fernandez. E. (2004), "A methodology for secure software design", *Proceedings of the International Conference on Software Engineering Research and Practice, SERP'04*, vol. 1, pp. 130-136.

Gilbert, P., Chun, B. G., Cox, L. P., & Jung, J. (2011). Vision: Automated security validation of mobile apps at app markets. *MobiSys'11 - Compilation Proceedings of the 9th Int. Conf. on Mobile Systems, Applications, and Services and Co-Located Workshops - 2011 Workshop on Mobile Cloud Computing and Services, MCS'11*, 21–25. https://doi.org/10.1145/1999732.1999740

Guzman, Aaron; Gupta, A. (2017). *IoT Penetration Testing Cookbook: Identify Vulnerabilities and Secure your Smart Devices*.

Huang, A. (2008). Similarity measures for text document clustering. *New Zealand Computer Science Research Student Conference, NZCSRSC 2008 - Proceedings*, (April), 49–56.

I. Revivo and O. Caspi. (2015). *Cuckoo-droid* (p. https://github.com/idanr1986/cuckoo-droid). p. https://github.com/idanr1986/cuckoo-droid.

ISCTE-IUL and Aptoide. (2019). AppSentinel. Retrieved January 11, 2020, from https://istar.iscte-iul.pt/news-posts/istar-iul-and-aptoide-join-efforts-to-combat-mobile-malware/

Ishii, Y., Watanabe, T., Kanei, F., Takata, Y., Shioji, E., Akiyama, M., … Mori, T. (2017). *Under-standing the Security Management of Global Third-Party Android Market-places*. *7*, 17. https://doi.org/10.1145/3121264.3121267

Jain, A. K., & Shanbhag, D. (2012). Addressing security and privacy risks in mobile applications. *IT Professional*, *14*(5), 28–33. https://doi.org/10.1109/MITP.2012.72

K. Peffers , T. Tuunanen, M. R. & S. C. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24:3*, 45–77. Retrieved from https://www.tandfonline.com/doi/abs/10.2753/MIS0742-1222240302

S. Seo, D. Lee and K. Yim. (2012). "Analysis on Maliciousness for Mobile Applications," *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Palermo, 2012, pp. 126-129, doi: 10.1109/IMIS.2012.190. https://ieeexplore.ieee.org/abstract/document/6296843

Lin, Y.-C. (2015). *Androbugs Framework* (p. https://github.com/AndroBugs/AndroBugs_Framework). p. https://github.com/AndroBugs/AndroBugs_Framework.

Malek, S., Esfahani, N., Kacem, T., Mahmood, R., Mirzaei, N., & Stavrou, A. (2012). A whitebox approach for automated security testing of Android applications on the cloud. *Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability Companion, SERE-C 2012*, 35–36. https://doi.org/10.1109/SERE-C.2012.39

N. Othman, R. F. and K. S. (2019). Manhattan Siamese LSTM for Question Retrieval in Community Question Answering. *The 18th International Conference on Ontologies, DataBases, and Applications of Semantics*.

NIAP, (2020). *National Information Assurance Partnership / Common Criteria Evaluation and Validation Scheme Publication# 2 Quality Manual and Standard Operating Procedures*. (January).

OWASP. (2011). OWASP Mobile Security Project.

OWASP. (2016). OWASP Mobile Mobile Top 10 (2016).

Ogata, M., Franklin, J., Voas, J., Sritapan, V., & Quirolgico, S. (2019). Vetting the Security of Mobile Applications. NIST Special Publication, 800–163(1). https://doi.org/10.6028/NIST.SP.800-163r1

Palmer, D. (2019). Mobile malware attacks are booming in 2019: These are the most common threats.

Pin, B., Salas, J., & Eguia, I. (2018). *Super Android Analyzer*. Retrieved from https://github.com/orgs/SUPERAndroidAnalyzer/people

Rastogi, V., Chen, Y., & Enck, W. (2013). AppsPlayground: Automatic security analysis of smartphone applications. *CODASPY 2013 - Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, 209–220. https://doi.org/10.1145/2435349.2435379

Shrivastava, A. (2017). *JAADAS*. Retrieved from https://github.com/flankerhqd/JAADAS

Shostack, A. (2018). The Threats To Our Products. *Microsoft SDL Blog*.

Stanek,M.(2017). "Secure by default - the case of TLS"

Vijayan, J. (2019). 32 application security stats that matter.

Wang, X., Shi, J., Yang, Y., Xu, K., Zeng, Y., & Tang, C. (2015). A novel hybrid mobile malware detection system integrating anomaly detection with misuse detection. *MCS 2015 - Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services, Co-Located with MobiCom 2015*, 15–22. https://doi.org/10.1145/2802130.2802132

Xanthopoulos, S., & Xinogalos, S. (2013). A comparative analysis of cross-platform development approaches for mobile applications. *ACM International Conference Proceeding Series*, 213–220. https://doi.org/10.1145/2490257.2490292

Velu, V, *Mobile Securty penetration Testing, Packt Publishing*, (2016)