Departamento de Ciências e Tecnologias da Informação

Sistema baseado na Web para Deteção e Apresentação de Vulnerabilidades em Aplicações Android

Raphael Alexandre Sperandio Candello

Dissertação submetida como requisito parcial para obtenção do grau de

Mestre em Software de Código Aberto

Orientador:

Doutor Carlos Serrão, Professor Auxiliar
ISCTE-IUL, Lisboa Outubro, 2019

Agradecimentos

Ao meu professor Carlos Serrão, que aceitou orientar-me neste projeto. À família, que apoia-me em momentos de estudo, em especial minha esposa que ajuda-me em todos os sentidos, minhas filhas que sabem que devo ter meu momento de tranquilidade e todos aqueles que encorajaram-me a sair do Brasil para tal projeto.

Aos colegas, sempre presentes em caso de necessidade, aos amigos que, mesmo longe, dão força e palavras de motivação.

A todos que aqui enumerei e a outros que apoiam-me de qualquer maneira, a minha gratidão!

Resumo

Com a crescente popularidade das plataformas móveis, mais utilizadores usam aplicações desenvolvidas por terceiros nos seus dispositivos. No entanto, algumas destas aplicações possuem vulnerabilidades que podem colocar os seus utilizadores e dados em risco. De igual forma, este tipo de plataformas está cada vez mais exposto a proliferação de *malware*.

Este trabalho tem o objetivo de apresentar uma ferramenta de análise de vulnerabilidades em aplicações Android, disponibilizada através de uma página *web*, que difere das ferramentas que oferecem funcionalidades semelhantes já existentes, englobando diversas ferramentas de deteção de vulnerabilidades, normalizando e categorizando as vulnerabilidades detectadas, e devolvendo ao utilizador um relatório de fácil compreensão que, não só apresenta as vulnerabilidades detectadas, mas aponta o que deve ser feito para resolvê-las.

A ferramenta é resultado de um estudo das aplicações de deteção de vulnerabilidades existentes, dos métodos de ataques de aplicações maliciosas sobre os dados do utilizador, e de uma forma de categorizar as vulnerabilidades para propor respostas as falhas de segurança.

Palavras-Chave: segurança; vulnerabilidade; Android; malware; OWASP; deteção.

Abstract

The mobile devices usage is rising, and with it the third-party applications installed on them, but these applications could have security vulnerabilities that put the user and their data in risk, therefore those devices are even more exposed to malware proliferation.

The objective of this paper is to introduce a vulnerability analysis tool over Android applications that is available through a website, which differs from other tools with similar purpose, because it includes several vulnerabilities detection tools and creates a single normalized report to expose software vulnerabilities and categorize them for user interpretation, adding guides to solve those security flaws over each vulnerability.

This solution is the result of a study over existent vulnerabilities detection tools, malicious software attack methods and software security flaws categorization to propose guides that could solve each of those vulnerabilities found.

Keywords: security; vulnerability; Android; malware; OWASP; detection.

Índice

Índice de Quadros

Índice de Figuras

Lista de Abreviaturas e Siglas

API – Application Programming Interface

CSS – Cascading Style Sheet

DOM – Document Object Model

HTML – Hypertext Markup Language

JSON – JavaScript Object Notation

JWT – JSON Web Token

OWASP – Open Web Application Security Project

Capítulo 1 – Introdução

Durante este capítulo será possível perceber a situação e definição do problema, a motivação do trabalho em questão, bem como a abordagem metodológica que foi seguida neste trabalho para apresentar uma proposta de resolução.

1.1. Enquadramento

Estima-se que, em 2019, existam mais de 7.7 bilhões de pessoas no mundo (Dadax, 2019), e 7.3 bilhões são utilizadores de dispositivos móveis (Statista, 2016), portanto aproximadamente 88% da população mundial possuí um aparelho móvel com aplicações instaladas para seu uso, com previsão de 94% da população mundial até o ano de 2023, conforme apresenta a *Figura 1*.

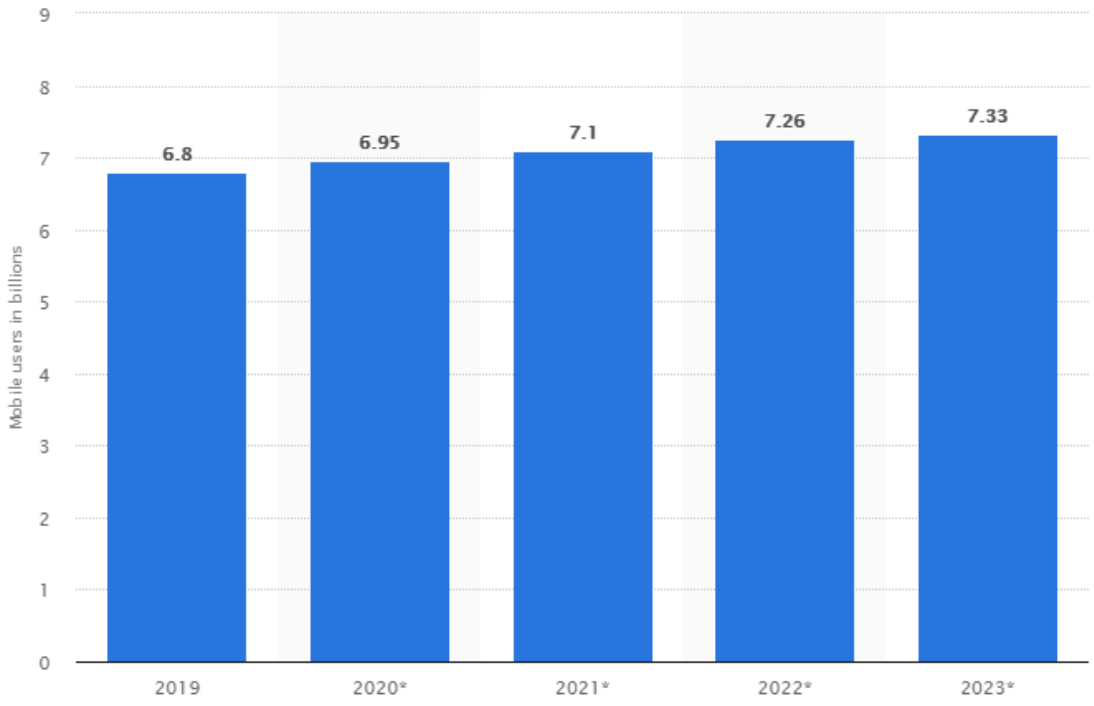

Figura 1. Utilizadores de aparelhos móveis no ano de 2019 com previsão a 2023. (Statista, 2019)

A utilização de um aparelho móvel, em 2019, sem aplicações não faz sentido, pois mesmo para simples chamadas de audio, é utilizado uma aplicação, que na grande maioria das vezes, é nativa do aparelho. Para desbloquear todo o potencial de *hardware* disponível nos aparelhos móveis, são disponibilizados, em lojas virtuais, milhares de aplicações, dos mais variados objetivos.

O número de *download* de aplicações móveis cresce a cada ano, como apresentado no gráfico da *Figura 2* (Statista, 2018), em torno de 195 bilhões de *downloads* em 2018 e estima-se 210 bilhões até o fim de 2019, portanto é possível dizer que, em média, cada pessoa no mundo faz o download de 27 aplicações por ano.

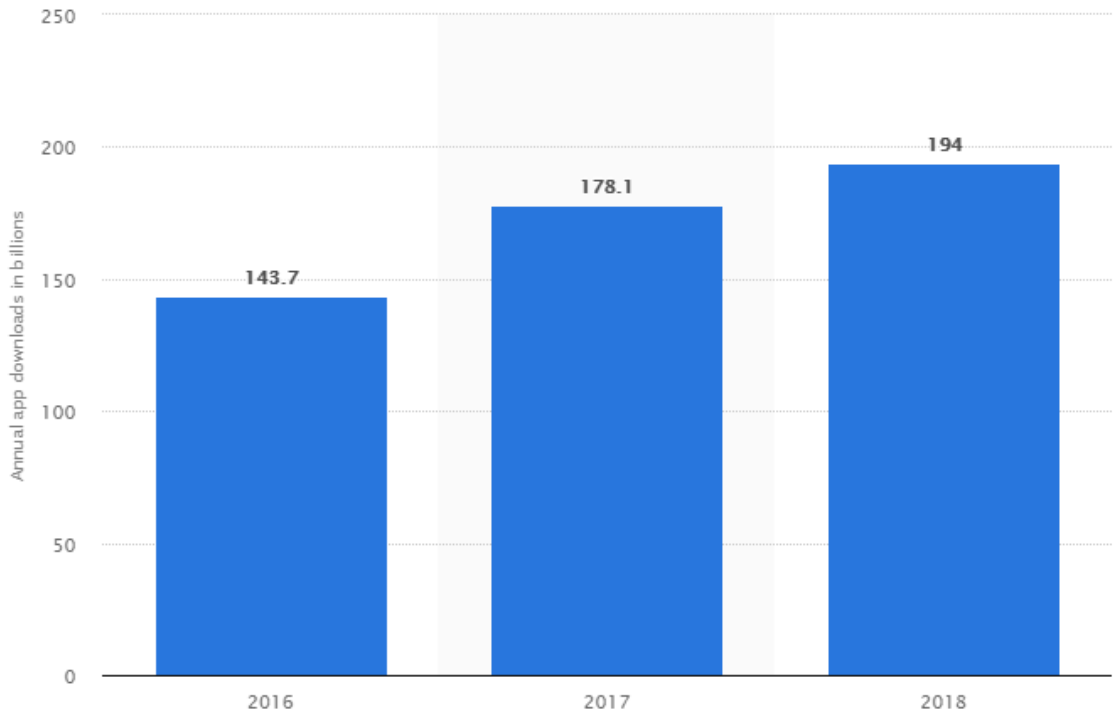

Figura 2. Download de aplicações móveis de 2016 a 2019 (Statista, 2019).

É indiscutível que as aplicações móveis estão fortemente presentes em 2019, e estas aplicações, que são dos mais variados tipos, para que possam servir de forma adequada, devem ter acesso aos mais diversos níveis de informação do utilizador que a carrega (J. Lin et al., 2012). Portanto conclui-se que o utilizador, ao fazer *download* da aplicação, confia que as aplicações escolhidas vão agir da forma esperada.

A segurança em dispositivos eletrónicos é, basicamente, a preocupação com a proteção dos dados que o utilizador tem armazenado em seu dispositivo, de forma consciente ou não, e o nível de vulnerabilidade dos dados depende do dispositivo em que se encontra (Botha, Furnell, & Clarke, 2009).

No período dos anos 80 iniciou-se uma preocupação em proteger os dados armazenados em dispositivos eletrónicos com a documentação da criação de uma aplicação que tinha como objetivo se replicar e espalhar-se por outros dispositivos (Robert & Chen, 2004). Este tipo específico de aplicação comumente contém rotinas de código que, copia seu código principal e se une a outras aplicações no sistema multiplicando-se o quanto conseguir, e outra rotina que contém seu objetivo, ou seja, um código que será executado e pode virtualmente fazer qualquer coisa, desde apagar ou alterar outros ficheiros, instalar novas aplicações ou enviar dados sensíveis para outro local (Cohen, Highland, & David, 1991). Por sua forma de se multiplicar como um vírus biológico, e de comumente ter intenções maliciosas, este tipo de aplicação foi denominado vírus de computadores (Robert & Chen, 2004).

Naturalmente foram criadas soluções para combater as aplicações que contém o comportamento de um vírus, e foram chamados de anti-vírus, e a luta contra esse tipo de aplicações tem vindo a ocorrer ao longo do tempo (Nachenberg, 1997).

Para serem infectados os dispositivos têm que entrar em contacto com o vírus de alguma forma, e dispositivos que estão praticamente permanentemente conectados a internet, como os *smartphones*, são muito mais sucetívies a ataques.

No caso específico dos *smartphones*, para instalar e executar uma aplicação, é necessário encontrá-la para *download*, comumente em uma loja virtual, e ter permissão do utilizador para iniciar, o que dificulta a proliferação de vírus, mas não diminui o risco de roubo de informações. No caso específico do Android, é igualmente possível a um utilizador instalar uma aplicação sem que a mesma seja proveniente de uma loja, o que aumenta o risco e exposição dos utilizadores. A *Figura 3* mostra que em 2004 foi detectado o primeiro vírus em dispositivos móveis, e com o passar dos anos, o número apenas aumentou (Hypponen, 2006).

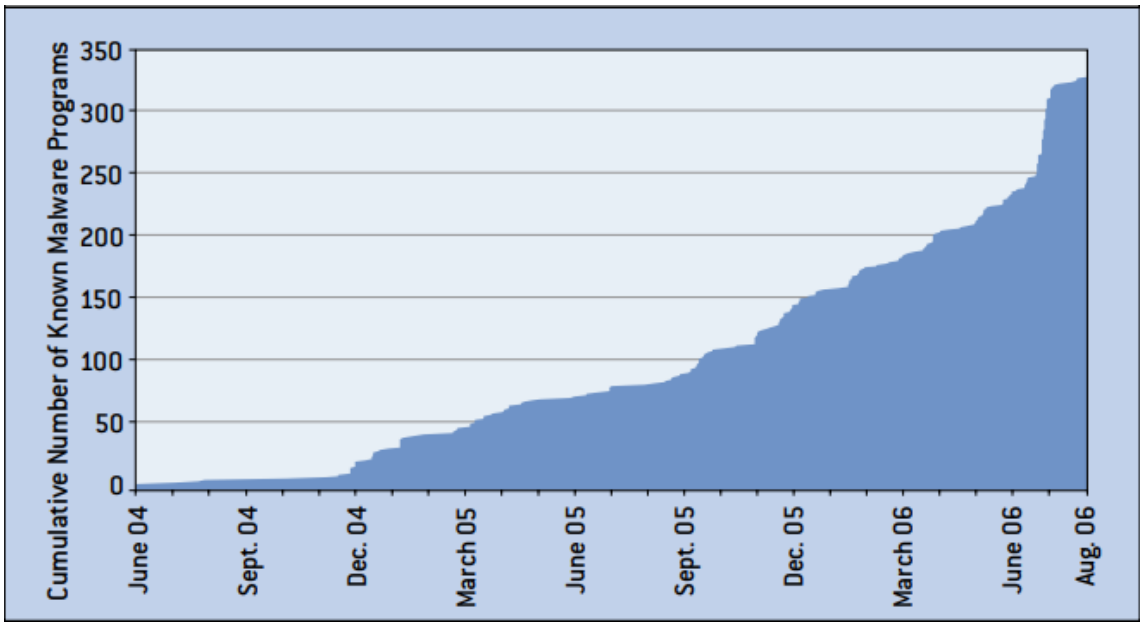

Figura 3. Número de malware encontrados em dispositivos móveis de 2004 a 2006 (Hypponen, 2006).

Na luta entre vírus e anti-vírus, as aplicações maliciosas alteraram a forma de se manifestar para infectar os dispositivos móveis, para que os anti-vírus não encontrem as rotinas comuns de infecção, estes *malwares* deixam de ser apenas rotinas de se auto-multiplicar e danificar, para serem aplicações completas que mascaram seu verdadeiro intuito (Narudin, Feizollah, Anuar, & Gani, 2016)

A tentativa destes *malwares* é de enganar o utilizador para que este permita sua entrada, e ainda, devem enganar a loja virtual que é a uma das maiores portas de entrada para o dispositivo móvel, dessa forma, as grandes lojas iniciam a batalha contra os *malwares*, mas infelimente não são tão eficazes quanto os utilizadores gostariam (Sanz et al., 2013).

Para além de todo este problema relacionado com o *malware* que pode afectar os dispositivos móveis e consequentemente os dados dos utilizadores, é importante considerar que a popularidade deste ecossistema móvel, essencialmente dominado pelo iOS e pelo Android (Statista, 2019), levou a que houvesse um crescimento muito significativo por parte dos programadores (quer a título individual ou corporativo) para oferecerem cada vez mais aplicações móveis. Infelizmente, muitas das práticas de desenvolvimento adoptadas por estes programadores não são as mais correctas em termos de segurança e consequentemente, acabam por deixar as suas aplicações vulneráveis, comprometendo o a segurança dos dispositivos móveis e dados dos utilizadores que as

instalam (McGraw, 2008). Por outro lado, as lojas de aplicações móveis apenas fazem um escrutínio simples sobre a possibilidade das aplicações terem vírus ou *malware* antes de disponibilizarem as mesmas a milhões de utilizadores, não realizando qualquer tipo de análise em termos de vulnerabilidades (Möller, Michahelles, Diewald, Roalter, & Kranz, 2012). Isto leva a que potencialmente um atacante possa explorar essas vulnerabilidades e comprometer o dispositivo e os dados de um utilizador.

Assim, uma das formas mais mais seguras de proteção contra *malware* é igualmente ter em consideração a segurança durante o desenvolvimento das aplicações, uma vez que boas práticas de desenvolvimento geram aplicações mais seguras, consequentemente protegidas contra possíveis ataques (Howard, 2004). Muitas das falhas de segurança em código são simples *bugs* (erros na codificação do aplicação) que podem ser facilmente encontrados com uma revisão do código, ou por ferramentas que façam esta análise de forma automática (McGraw, 2008).

Como a revisão de código deve ser feita por outro programador, e consome muito tempo além de ser desgastante para quem faz a revisão, foi natural a criação de ferramentas que pudessem fazê-la de forma automática, que começaram a aparecer por volta do ano 2000, e com mais força em 2004 (McGraw, 2008).

1.2. O Problema

De acordo com o enquadramento da situação na seção anterior deste capítulo e com a imensa quantidade de informações em *smartphones* que estão conectados a internet quase todo o tempo, como é possível travar uma luta eficiente contra vulnerabilidades nas aplicações e contra os *malwares,* e assim manter o dispositivo móvel e os dados do utilizador em segurança?

Tendo em consideração que a grande porta de entrada para as aplicações num dispositivo móvel são as lojas virtuais, estas devem ser o primeiro ponto bloqueio contra a aplicação maliciosa e/ou vulnerável. Como as lojas têm interesse em manter os utilizadores fiéis às mesmas, é natural a preocupação em criar um bloqueio efetivo para não perder clientes.

As grandes lojas de aplicações móveis têm vindo a tomar medidas para impedir a proliferação de *malwares*, em julho de 2009 a Apple apresentou um rigoroso sistema de revisão do código enviado para loja (Sanz et al., 2013), e a Google utiliza um sistema em

que o utilizador deve dar permissão para cada acesso da aplicação, e ainda, as duas lojas tem cláusulas nos termos de serviços contra o envio de rotinas maliciosas de código e sistemas de anti-vírus para eliminar *malwares* que ultrapassem as barreiras anteriores. Infelizmente essas medidas no geral são ineficazes (Sanz et al., 2013), nenhuma das duas plataformas oferece formas eficientes de controlar as vulnerabilidades de desenvolvimento das aplicações móveis.

Para além das aplicações que foram criadas para causarem mal ao utilizador, existe ainda o problema da aplicação desenvolvida com falhas, que são aplicações legítimas sem o intuito malicioso, mas que por terem um código fraco em termos de segurança, deixam brechas para ataques externos de *hackers* ou de outras aplicações que são maliciosas, o que causa o mesmo problema as lojas virtuais e, consequentemente, ao utilizador final (McGraw, 2002).

Portanto é possível notar que na atual situação o mesmo problema partilhado em dois pontos de vista, do lado do utilizador final a preocupação de como manter seus dados em segurança, e do lado do programador da aplicação, como criar uma aplicação que não deixe o utilizador vulnerável a ataques.

A solução que este trabalho sugere é uma ferramenta que possa analisar as aplicações e devolver como resposta as vulnerabilidades ou ameaças encontradas, e ainda, uma forma de resolvê-las. Esta ferramenta pode ser usada pelo programador da aplicação, para avaliar seu trabalho e enviar as lojas uma aplicação com qualidade de código e livre de rotinas maliciosas ou brechas para entradas de *hackers*. Pode também ser usada pelas lojas virtuais para avaliar as aplicações de forma mais ágil e poder eliminar as aplicações que não atenderem aos requisitos de segurança, ou ainda, negar a entrada da aplicação submetida pelo programador. E ainda pode ser utilizada pelo utilizador final que pode verificar as vulnerabilidades de uma aplicação antes de instalar em seu dispositivo.

Este sistema sugerido para solucionar o problema proposto tem a intenção de utilizar ferramentas existentes de análise de aplicações móveis em sistemas Android, obter o resultado de cada uma dessas análises, normalizar e categorizar a resposta, e retornar, de uma forma agradável a visualização, os problemas e sugestões de solução, e a partir deste momento será referida neste trabalho como DAAV (*Detection of Android Aplication Vulnerabilities*).

1.3. Objetivos da Tese

Este trabalho tem como objetivo estudar as ferramentas já existentes de detecção de vulnerabilidades em aplicações móveis de sistema Android e implementar um protótipo de um sistema *web* que normalize as repostas das ferramentas estudadas anterioremente e retorne ao utilizador, de uma forma mais clara, as vulnerabilidades encontradas e como é possível corrigi-las para ter uma aplicação segura.

Durante o estudo das ferramentas existentes, terá prioridade a verificação de suas funcionalidades e formas de apresentação de resultado, verificar a praticidade de seu uso, linguagem de programação utilizada e por fim, a possibilidade de pequenas alterações para integração em um sistema maior como o DAAV.

Em seguida um estudo sobre a melhor forma de categorizar as diferentes respostas encontradas e assim normalizar os resultados para apresentação de uma única análise de vulnerabilidade, que possa conter indicações de como resolvê-las.

Com as ferramentas escolhidas e uma única forma de categorizar as respostas obtidas, cria-se a estrutura do sistema sugerido, com armazenamento dos resultados para então apresentação ao utilizador, e por fim a implementação do protótipo DAAV em *web*.

1.4. Questões de Investigação

A questão de investigação que este trabalho se propõe a responder é referente a segurança do utilizador de um dispositivo móvel com aplicações de sistema Android, envolve a luta entre aplicações legítimas e *malwares*, e a vulnerabilidade dos dados do utilizador neste dispositivo.

Ao levar em consideração que a melhor forma de se proteger contra estes ataques é o desenvolvimento de aplicações seguras e, portanto, que se preocupa com as boas práticas de segurança no código, é possível desenvolver uma ferramenta automática de análise de vulnerabilidades para a *web* que ajude os programadores a identificar as vulnerabilidades das suas aplicações e oferecer *feedback* sobre a forma como mitgar as mesmas com o objetivo de melhorar a segurança geral das aplicações?

Existem múltiplas ferramentas de análise de vulnerabilidades que estão disponíveis para os programadores, a grande maioria indica o trecho de código que possa conter falhas de segurança, mas nenhuma apresenta sugestões sobre a forma como mitigar estas vulnerabilidades. O protótipo desenvolvido (DAAV) propõe-se a responder esta pergunta, ao apresentar ao programador, não somente as possíveis falhas de código, mas uma indicação de como solucioná-las.

1.5. Metodologia de Pesquisa

As metodologias de pesquisa consistem em um conjunto de procedimentos e regras aceitos pela comunidade académica que ajudam a construção de um conhecimento científico (Lacerda, Dresch, Antunes Júnior, & Proença, 2013) e para este trabalho foi escolhido a metodologia Design Science Research que baseia-se na criação de um artefacto, ou protótipo de um artefacto em teoria, que traga melhorias ao atual estado da arte, com pesquisa do conhecimento existente (Kuechler & Petter, 2012).

Hevner (Hevner, March, Park, & Ram, 2004) propõe em sua pesquisa, sete guias que devem ser cumpridos no uso da Design Science Research como metodologia de pesquisa, que estão caracterizados na *Tabela 1*.

1	Criação de um artefacto	A criação de um artefacto é o resultado da pesquisa que soluciona os problemas propostos, deve ser descrito de forma eficiente incluindo forma de implementação e aplicação.
2	Relevância do problema	Obtenção de conhecimento que comprova a relevância de um problema e necessidade da criação de uma solução.
3	Validação do artefacto	Comprovar a eficácia do artefacto se este obtiver os requerimentos necessários, como por exemplo um estudo de caso.

4	Contribuição de pesquisa	O resultado deve ser algo novo que contribui para a área de pesquisa proposta, ou seja, solucionar os problemas propostos.
5	Rigor da pesquisa	Obter dados para criar o artefacto de forma apropriada, evitar informações ou dados duvidosos.
6	Métodos de busca	Criar um método de busca de conhecimento que encontre a solução de forma efetiva, percebe-se o melhor método de busca quando o resultado demonstra a relevância do artefacto.
7	Comunicação da pesquisa	A pesquisa deve ser apresentada de forma técnica, ao explicar sua criação e implementação, e de forma administrativa, ao apresentar sua relevância em um contexto organizacional. De forma a pesquisadores perceberem como aprimorar o artefacto e utilizadores percebam sua importância prática de uso.

Tabela 1. Guias propostos por Hevner para Design Science Research

1.6. Estrutura da Tese

Este estudo está organizado em cinco capítulos que contemplam introdução, revisão da literatura, desenvolvimento da solução, análise de resultados e conclusão.

O primeiro capítulo apresenta o enquadramento do problema, o problema em si, a motivação e objetivos deste trabalho e as questões e metodologia de investigação. Segue o segundo capítulo com a revisão bibliográfica e como se encontra o estado da arte, um estudo do ambiente em que se apresenta o problema e as ferramentas existentes na tentativa de solucioná-lo.

O terceiro capítulo tem como objetivo expor a estrutura do artefacto proposto para solucionar o problema apresentado no segundo capítulo, a detalhar seus requisitos, arquitetura, diagrama de sequência de métodos e resultados fornecidos pelo artefacto.

O quarto capítulo expõe um estudo de caso, a exemplificar o uso do artefacto, suas funcionalidades, informações e forma de apresentação. E por fim, o quinto capítulo apresenta as conclusões desta tese, o resultado criado pelo artefacto proposto e apresentação de possíveis trabalhos futuros.

Capítulo 2 – Revisão bibliográfica

Este capítulo apresenta o estado da arte através de uma pesquisa científica e transcrição de informações relevantes ao ambiente que se encontra o problema, e por fim o caminho para a solução proposta.

2.1. Vulnerabilidades em sistemas Android

Aplicação maliciosa, ou *malware*, é uma aplicação desenvolvida para atacar outras aplicações, simplesmente danificando-as, ou fazendo com que a aplicação legítima se comporte de uma forma diferente do esperado (Kramer & Bradfield, 2010). Além disso, podem ser considerados *malware* as aplicações criadas para mascarar seu verdadeiro propósito, que por fim, agem de forma inesperada pelo utilizador e tem o mesmo objetivo de danificar ou alterar outros componentes de uma aplicação ou sistema, comumente tem o objetivo final de roubar informações sensíveis do utilizador, ou abrir caminhos para que isso seja possível (Lopes, Serrão, Nunes, Almeida, & Oliveira, 2019).

A história documentada de *malwares* sobre sistema Android tem seu primeiro *malware* datado de 2010 (Zhou & Jiang, 2012), e desde então foram detectadas algumas formas padrão de infecção, dentre elas a mais comum é o *repackaging*, neste caso o programador do *malware* se apodera de uma aplicação legítima, altera o pacote da aplicação para incluir o *malware*, recria o pacote com o *malware* incluído, e volta a disponibiliza-la em lojas de aplicações para que os utilizadores façam *download*, neste caso a aplicação nova está mascarada como uma aplicação legítima já existente no mercado, mas com o pacote de infecção incluso (Zhou & Jiang, 2012).

Update attack é uma forma mais complexa de infecção, mas que parte do princípio do *repackaging*, pois o deselvolvedor do *malware* altera uma aplicação legítima, mas ao invés de aplicar todo o conteúdo do *malware* diretamente sobre a aplicação, inclui apenas uma rotina de código para futuramente fazer *download* do *malware* por completo (Tenenboim-Chekina et al., 2013). Esse método torna este *malware* de difícil detecção, porque as lojas ao fazer busca por pacotes maliciosos de dados não os encontram, pois neste ataque o conteúdo incluído foi apenas uma rotina de *download*, o restante do pacote

que seria de fácil detecção, será recebido pelo utilizador após execução da aplicação (Zhou & Jiang, 2012).

Outra forma comum de infecção é o que podemos chamar de *drive-by download*, que ao invés de alterar uma aplicação, o próprio *malware* se faz parecer uma aplicação legítima e tenta enganar o utilizador a fazer *download* e executá-la (Cova, Kruegel, & Vigna, 2010). Nesta categoria existem inúmeras formas e tipos de infecção, desde aplicações que apenas aparentam dar algum resultado esperado pelo utilizador, com o intuito de ser executado, e então, o *malware* ataca imediatamente, até aquelas que são efetivamente uma aplicação, mas que fazem mais do que aparentam e, de forma sorrateira, alteram, roubam, acessam ou fornecem acesso a dados sensíveis do utilizador (Zhou & Jiang, 2012).

Aplicações legítimas que não sigam as boas práticas de criação de código, ou programadores que simplesmente não tem preocupação com segurança durante seu desenvolvimento, deixam a aplicação exposta a vulnerabilidades que, tornam-se alvo de *malwares* e consequentemente são perigosos para o utilizador (McGraw, 2002).

O departamento de segurança interna norte-americano (CISA) (U.S. Department of Homeland Security, 2019) que analisa ciber-segurança e reporta boletins atualizados de falhas de código, apresenta desde o ano de 2004, que em torno de 92% dos incidentes que envolvem computadores são provenientes de falhas de código, portanto descaso com a segurança no desenvolvimento, erros de implementação ou deficiência na codificação devem ser supridos para evitar ataques (Sotirov, 2005).

A detecção de *malwares* se baseia praticamente no comportamento das aplicações, ou seja, aplicações que tentam aceder a eventos ou recursos específicos do sistema Android tem maior chance de ter intenções maliciosas. Eventos como aumentar o nível de privilégio da aplicação, permitir acesso remoto ou acesso a dados do utilizador sem a permissão do mesmo podem ser facilmente detectados como comportamentos de *malwares* (Zhou & Jiang, 2012).

Eventos como *download*, leitura e escrita de ficheiros são comuns em todas as aplicações, e se monitorados de forma isolada, fica impossível separar um aplicação legítima de um *malware*, mas se forem analisadas de forma temporal, nota-se o ciclo de vida da grande maioria dos *malwares* (Bose, Hu, Shin, & Park, 2008). O evento de *download* é muito comum em aplicações móveis, os recursos de ficheiros são comumente

acedidos durante qualquer execução, mas a sequência de um *download* de um ficheiro de configuração, tentativa de armazenamento e execução, e sobrescrita de permissões do sistema, é claramente o ciclo de vida de um *malware*, enquanto esses eventos isolados podem ser de uma aplicação legítima.

A análise estática é do tipo que busca no código da aplicação por eventos específicos, ou por uma sequência desses eventos, sem a necessidade de executar a aplicação suspeita, desta forma serve como filtro e elimina grande parte das ameaças antes de chegar ao utilizador final. A análise dinâmica necessita a execução da aplicação em um ambiente de testes, e tem como objetivo monitorar o comportamento em busca de movimentações suspeitas, o problema deste tipo de análise é não saber ao certo quanto tempo deve-se analisar a aplicação, e quais são os requerimentos para iniciar o comportamento malicioso (Chandramohan & Tan, 2012).

A preocupação com segurança durante o desenvolvimento do aplicação é essencial para evitar incidentes aos dados do utilizador, mas seguir as boas práticas de código e revisão do que foi criado é um trabalho difícil e que consome muito tempo (McGraw, 2008), apesar de existirem ferramentas que ajudam nas análises, estas são de difícil utilização pelo programador e não trazem uma resposta direta de como solucionar as possíveis falhas encontradas. Como normalmente existe uma maior pressão para o desenvolvimento das aplicações, e como a maioria dos programadores têm poucos conhecimentos de segurança, as aplicações desenvolvidas carecem de boas práticas de segurança que se traduzem em possíveis vulnerabilidades. No entanto, existem organizações, como por exemplo a OWASP (*Open Web Applications Security Project*), que desenvolvem um conjunto de projetos que visam melhorar a segurança das práticas de desenvolvimento dos programadores.

2.2. OWASP

O *Open Web Application Security Project*, ou simplesmente OWASP foi criado a partir de uma lista de emails por Mark Curphey em 2001 com o objetivo de documentar e compartilhar conhecimento sobre segurança em aplicações *web* (Blyth, 2004), e desde a criação deste documento, é uma organização sem fins lucrativos, que estimula um conjunto de projetos em regime aberto (quer de aplicações de *software*, e de

documentação) para a partilha e disseminação de conhecimento relacionado com segurança aplicacional. No site *web* da OWASP é possível ter acesso a inúmeros projetos, guias e publicações sobre segurança aplicacional. Um dos projetos mais representativos da OWASP é o OWASP Top Ten, que lista os 10 principais riscos relacionados com o desenvolvimento aplicacional em segurança.

Como referido, dentre os projetos da comunidade, OWASP Top Ten publicado em 2010 tornou-se referência a todos os programadores de aplicações como um guia na tentativa de criação de uma aplicação segura, pois descreve os riscos de segurança mais comuns, agrupa por nível de severidade e os apresenta em um ranking dos dez maiores grupos (OWASP Foundation, 2015).

Em 2011 foi criado igualmente o OWASP Mobile Top Ten, com o mesmo intuito do projeto OWASP Top Ten, mas focado em aplicações móveis (OWASP, 2014). É uma classificação importante para identificação de vulnerabilidades e para padronização dos resultados encontrados por diversas ferramentas existentes de detecção de falhas de segurança. Apesar da última versão oficialmente publicada ser datada de 2016, a classificação é pertinente em 2019 e tem a nomenclatura de M1 a M10 como descrita na *Tabela 2*.

M1 – Utilização inapropriada da plataforma	A plataforma de desenvolvimento de código já inclui controle de segurança, a má utilização ou descaso a estes controles permanecem nesta categoria.
M2 – Falha de segurança em armazenamento de dados	Assumir que os dados internos não são acessíveis, como sistema de ficheiros e base de dados.
M3 – Falha de segurança na comunicação e troca de dados	Sistema de comunicação fraco, desde abertura de sessão cliente e servidor a envio de dados.
M4 – Falha de segurança na autenticação do utilizador	Desde falha ou falta de dados para autenticar o utilizador legítmo a fraca manipulação da sessão.

M5 – Criptografia insuficiente	Nesta categoria não se aplica descaso a criptografia, mas sim quando se tenta aplicá-la de forma falha ou insuficiente.
M6 – Autorização não segura	Falha nas decisões de autorização, como autorizar utilizadores não permitidos.
M7 – Má qualidade de código	Qualquer falha no código do dipositivo móvel (cliente), que a solução seja re-escrever o código.
M8 – Violação de código	Permitir a alteração do código já implementado no cliente.
M9 – Engenharia reversa	Permitir acesso a informações sensíveis do código através do uso de engenharia reversa.
M10 – Funcionalidade não desejada	Falha do programador a remover funções ou informações que servem durante o desenvolvimento mas não devem estar no produto final.

Tabela 2. Classificação OWASP Top Tem Mobile de falhas de segurança.

2.2.1. M1 – Utilização inapropriada da plataforma

A plataforma de desenvolvimento da aplicação, como por exemplo Android ou iOS, já contém diversos controles de segurança que devem ser aplicados pelo programador, portanto qualquer falha ou má utilização destes componentes de segurança é adicionado a classificação M1 da OWASP.

Keychain é um sistema de armazenamento seguro de informações sensíveis do desenvolvimento de uma aplicação em iOS, a utilização de qualquer outro sistema para armazenar esses dados que não seja o *Keychain* é uma falha de segurança M1, pois a própria plataforma disponibiliza este serviço como seguro.

Nos casos de falha M1 é comum os ataques que buscam por chamadas a serviços e a tentativa de incluir código malicioso no retorno destas chamadas, causando comportamentos inesperados da aplicação legítima.

2.2.2. M2 – Falha de segurança em armazenamento de dados

Este tipo de falha acontece quando o programador assume que não é possível o acesso ao sistema de ficheiros e dados sensíveis de seu desenvolvimento, o que não é verdade, pois em posse de um dispositivo móvel o sistema de ficheiros pode ser facilmente acessado.

O programador deve conhecer estes métodos de acesso e proteger seus dados sensíveis, como por exemplo com criptografia. Este tipo de falha pode levar facilmente ao método de *repackaging* e incluir um *malware* na aplicação legítima, além de imediatamente aceder aos dados sensíveis da aplicação, comprometendo não apenas um, mas todos os utilizadores.

2.2.3. M3 – Falha de segurança na comunicação e troca de dados

Praticamente todas as aplicações comunicam-se com um servidor para troca de dados com o cliente, o monitoramento do tráfico de informações pode apanhar uma comunicação com falha de segurança e obter acesso aos dados enquanto transita pela rede, essa falha é classificada como M3 pela OWASP.

O uso de um transporte de dados seguro pela aplicação não significa que esta a tenha utilizado de forma correta, monitorar o tráfico de informações de uma rede pode facilmente detectar estas falhas e disponibilizar dados sensíveis da aplicação, e a troca de informações não precisa ser necessariamente de cliente para servidor, pode ser entre dois dispositivos móveis.

2.2.4. M4 – Falha de segurança na autenticação do utilizador

Autenticações fracas, ou inexistência delas, faz com que um utilizador qualquer possa aceder a funções ou informações de forma anônima, permitindo assim acesso não apropriado.

Aplicações *web* tem tradicionalmente uma forte comunicação com o servidor, e, portanto, podem depois de autenticados, manter a sessão do utilizador legítimo aberta durante o uso. No caso de dispositivos móveis não se espera que o utilizador esteja ativo

durante toda a sessão, devido a forma imprevisível de estabilidade da conexão, para solucionar esse problema é comum as aplicações móveis fazerem autenticações no próprio dispositivo sem contacto com o servidor. Se houver uma altenticação local fraca, um utilizador pode falsificar sua entrada e alterar a sessão de forma que, ao conectar novamente ao servidor, este pense que permanece em uma conexão legítima.

2.2.5. M5 – Criptografia insuficiente

Para estar nesta categoria, a aplicação deve utilizar de criptografia em seus dados sensíveis, mas de forma ineficiente, tornando-se alvo de utilizadores que possam reverter a criptografia e assim aceder aos dados sensíveis da aplicação.

Escolher um método de criptografia descontinuado, utilizar de seu próprio método de criptografia, ou utilizar uma excelente ferramenta criptográfica de forma errada disponibilizando de alguma forma a chave criptográfica, são exemplos de vulnerabilidades que tem categoria M5 na OWASP.

2.2.6. M6 – Autorização não segura

Diferente da categoria M4, o utilizador que ataca não tenta ser autenticado pelo sistema, este já foi autenticado, mas tenta ter um nível de privilégios maior do que realmente deve ter, ou seja, tenta aceder a funções que não tem autorização.

A aplicação deve conter fortes procedimentos de autorização para cada execução de funções ou chamadas ao servidor, principalmente enquanto tem a sessão aberta, mas não estão em contato com o servidor, conforme descrito no ítem 2.2.4 deste capítulo. A falha ou falta de autorização do utilizador pode permitir o acesso de dados que este não deveria ter, ou abrir acesso a outras funções a qual não deveria, e a melhor forma de evitar isso é uma verificação de autorização no servidor mesmo depois de avaliada pelo dispositivo.

2.2.7. M7 – Má qualidade de código

Código de má qualidade, mesmo que em áreas da aplicação que não envolvam dados sensíveis, podem levar a falhas graves de segurança, e comumente iniciam-se em partes da aplicação que contém entrada de dados pelo utilizador.

Ao seguir as boas práticas de código o programador protege seu sistema contra ataques desta categoria, principalmente nas entradas de dados dos utilizadores, caso estas informações não sejam avaliadas no código, pode ser possível incluir código malicioso por estes meios e o sistema deixa de agir como previsto. É comum em aplicações, o campo destinado a preenchimento da morada não ter avaliação alguma de entrada de dados, porque difere muito de um país para outro e pode conter os mais diferentes caracteres e formatos, dessa forma é uma porta de entrada para adicionar código que possa alterar a funcionalidade do sistema.

2.2.8. M8 – Violação de código

Esta classificação baseia-se no *repackaging*, ou seja, são aplicações que foram alteradas com inclusão de código malicioso e retornadas as lojas na tentativa de parecer com a aplicação legítima. Uma vez instalado no dispositivo móvel o código pode ter todal acesso e funções que precisar, pois o código malicioso já foi instalado e executado com persmissão do utilizador.

A única forma de prevenir a violação de código, é a aplicação validar durante seu uso se houve alterações em seu código e assim impedir sua utilização, e ainda, a aplicação deve detectar se a plataforma em que foi instalada não está comprometida, como por exemplo o *root* em Android ou *jailbreak* no iOS.

2.2.9. M9 – Engenharia reversa

No geral qualquer aplicação está sujeita a engenharia reversa, afinal é impossível impedir um utilizador de fazer *download* da aplicação e aplicar ferramentas externas de análise de código. O intuito do utilizador malicioso é perceber o funcionamento da aplicação, variáveis utilizadas e chamadas a funções para então localizar alguma falha para explorar.

Algumas linguagens de programação que permitem alterações dinânimcas são mais suscetíveis a engenharia reversa, mas os programadores não devem se limitar a escolha de linguagem para impedir este tipo de ataque, uma ferramenta de ofuscação é a melhor solução para este caso. Ferramentas de ofuscação são aplicações criadas para proteção de

código, que basicamente, transforma o código em um equivalente, mas de difícil leitura e aplicação de engenharia reversa (Collberg & Thomborson, 2002).

2.2.10. M10 – Funcionalidade não desejada

Neste caso a aplicação deve conter alguma funcionalidade que está escondida a interface do utilizador, esta pode ter sido deixada propositalmente pelo programador ou pode ser uma função que deveria ser utilizada enquanto se desenvolve, e por acidente, foi deixada na aplicação.

É comum durante a criação, o programador ter ficheiros de registo de informações, ou funções de teste, variáveis com senhas ou acesso a servidores restritos para acelerar o processo de desenvolvimento. O que não pode ser feito de forma alguma é disponibilizar essas funções no produto final entregue as lojas de aplicações, e a melhor forma de se evitar esse tipo de ameaça é uma revisão de código por outro programador.

2.3. Ferramentas de análise de aplicações móveis

Diversas ferramentas foram criadas na tentativa de detectar vulnerabilidades em aplicações de sistema Android, idealmente estas ferramentas devem apontar com precisão todas as falhas de segurança, mas a grande parte das opções encontradas apenas apontam possíveis ameaças em partes do código, e é dever do programador procurar por estes trechos de código e verificar se há realmente uma falha de segurança para correção, e ainda, descobrir como aplicar tal correção.

Neste sub-capítulo serão descritas algumas ferramentas de análise estática de código criadas em *python*, todas requerem conhecimento de desenvolvimento de aplicações, e não são simples de usar ou configurar. Todas as ferramentas aqui descritas são de código aberto, e foram escolhidas dessa forma porque são frequentemente atualizadas, e a disponibilidade do seu código-fonte e documentação permite a realização de alterações para inclusão noutros sistemas de acordo com o seu modelo de licenciamento.

2.3.1. Androguard

É uma ferramenta criada por Anthony Desnos que reverte aplicações Android de ficheiros APK, e além de fazer uma análise de vulnerabilidades (Desnos & Gueguen, 2011), possibilita interagir diretamente com o código malicioso, verificar seus eventos, recursos e comparar com outras vulnerabilidades (Dunham, Hartman, Morales, Quintans, & Strazzere, 2014). Por ser uma ferramenta com diversas funcionalidades, muitas outras ferramentas em *python* utilizam do Androguard como base para seu funcionamento.

Apesar de ser muito poderosa, esta ferramenta não tem um resultado simples. É possível analisar, comparar e interagir com código malicioso, mas não resulta em um documento ou resposta direta a quem utiliza, é preciso maiores análises sobre os possíveis retornos do Androguard para chegar a qualquer conclusão sobre vulnerabilidade no código.

2.3.2. ApkTool

Criado originalmente por Ryszard Wiśniewski e disponibilizado ao público em 2010, esta ferramenta utiliza métodos de engenharia reversa em aplicações Android já fechadas em binário e reconstrói seu código praticamente como o original, para que seja possível análise em separado de trechos de código, métodos e ficheiros. Pode ser utilizada para alterar o código original, com intenção de incluir novas funcionalidades e correções, e depois recriar a aplicação para futura utilização, ou seja, permite o *repackaging*, e por isso o criador deixa em sua documentação oficial de forma clara que sua ferramenta não deve ser usada para inclusão de *malware* em aplicações legítimas, apesar de não conseguir impedir que aconteça (ibotpeaches, 2016).

É uma ferramenta muito poderosa nas mãos dos programadores de aplicações para Android, mas por não ser simples de utilizar e não devolver nenhum resultado, análise ou relatório, tem pouca utilidade para o utilizador final.

2.3.3. Androbugs

Baseado no Androguard para reverter ficheiros APK, Androbugs foi criado por Yu-Cheng Lin com o intuito de devolver ao utilizador um relatório da análise de vulnerabilidade, é eficiente e criado para análises massivas de APKs. Busca por

comandos perigosos, más práticas de código, trechos de código que são conhecidos como vulneráveis a ataques, e ainda verifica se a aplicação foi alterada por *repackaging* (Y. Lin, 2015).

O programador do Androbugs preocupou-se com o tempo de análise do APK durante sua criação, para que pudesse ser uma ferramenta de análise massiva, e foi bem sucedido, pois a ferramenta não analisa todas as linhas de código uma a uma, mas percorre o código em busca de funções potencialmente perigosas e se encontra algo malicioso, volta a esta função e então percorre o código, dessa forma o Androbugs não demora mais de um minuto para retornar o relatório de vulnerabilidades.

Apesar de não ter uma interface amigável para o retorno do resultado ao utilizador, o Androbugs cria um relatório completo, que contém o trecho de código com falha de segurança, o nível de severidade da falha separado por categoria conforme a *Tabela 3*, recomendações para mitigar o problema e detalhes conhecidos sobre a vulnerabilidade encontrada.

Critical	Vulnerabilidade confirmada que precisa ser resolvida.
Warning	Necessidade de confirmação manual do utilizador.
Notice	Vulnerabilidade de baixa prioridade.
Info	Nenhuma vulnerabilidade encontrada.

Tabela 3. Classificação de nível de severidade do Androbugs.

2.3.4. CuckooDroid Sandbox

Criado em 2010 por Claudio Guarnieri, esta ferramenta faz, não apenas análise estática, mas também dinâmica, a executar a aplicação que será testada em um ambiente seguro para não afetar a máquina principal (Sandbox, 2011). Não é uma ferramenta amigável para o utilizador final e depende de muito tempo e conhecimento para ser configurada, não apenas a máquina que será designada servidor da ferramenta, que contém uma API local para submeter a análise e recuperar o resultado, como as máquinas virtuais que são criadas para execução segura da aplicação móvel a ser analisada.

CuckooDroid Sandbox foi criado para ser uma ferramenta autossuficiente, mas também para ser integrada em outros sistemas, por isso utiliza do método de entrega de dados e retorno de respostas da sua API interna, dessa forma outras aplicações podem efetuar os pedidos a API da CuckooDroid Sandbox.

Apesar de apresentar o retorno da análise em diversos formatos, inclusive em objecto JSON (JavaScript Object Notation), a ferramenta não foi criada para análise específica de ficheiros APK, e, portanto, faz uma busca por vulnerabilidades através de comparações com conteúdos maliciosos já categorizados em sua API. As análises e resultados podem ser configuradas como todo o restante da ferramenta, neste caso é possível incluir análises específicas de aplicações Android, mas como tem grande poder de customização, a ferramenta retorna resultados de difícil normalização e categorização de vulnerabilidades.

2.3.5. DroidStatX

É uma ferramenta de análise estática e dinâmica criada em 2015 por Claudio André, que está a receber atualizações constantes, e tem o objetivo de não apenas descrever as vulnerabilidades, mas apresenta-las de forma amigável e ainda disponibilizar uma metodologia de trabalho para corrigi-las (André, 2019).

Baseado na ferramenta Androguard e ApkTool, e com a utilização da aplicação XMind, cria como resultado da análise um mapa mental, como o exemplo da *Figura 4*, de fácil percepção e baseado na categorização da OWASP para classificar as vulberabilidades, e ainda, apresenta *links* com sugestões de solução para cada uma das falhas de segurança.

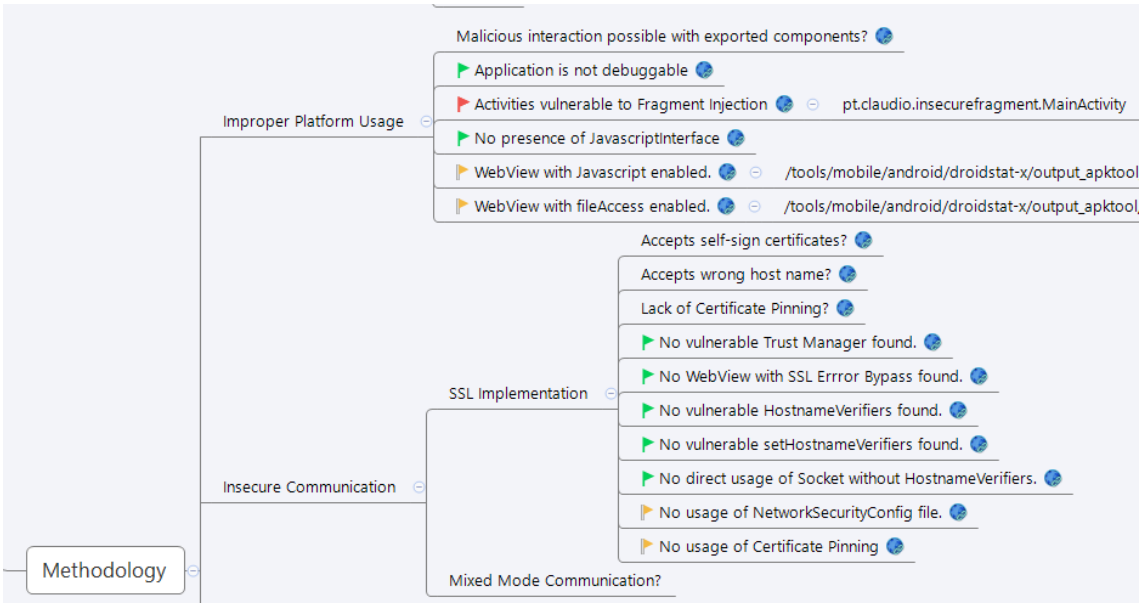

Figura 4. Trecho de exemplo da estrutura do mapa mental criado pelo DroidStatX (André, 2019)

Pode ser considerada a ferramenta mais próxima da solução proposta por essa tese, pois analisa aplicações e retorna um resultado de interpretação simples a programadores, porém limita-se apenas a uma análise, não é escalonável e não é simples de utilizar ou configurar.

2.3.6. AndroWarn

Criado por Thomas Debize em 2013, recebe atualizações constantes até o presente momento, e é considerado pelo próprio autor "mais uma ferramenta de análise estática de vulnerabilidades em aplicações Android" (Debize, 2013). Baseia-se na ferramenta Androguard e procura especificamente por doze tipos de comportamentos que podem ser considerados maliciosos: extração de informação sensível do dispositivo, extração de informação do utilizador, extração de informação da localização geográfica do dispositivo, extração de informação de conecções do dispositivo, abuso no uso de serviços de comunicação, métodos de conexão remota, intercepção de dados de áudio e vídeo, envio de dados sensíveis, acesso a memória externa, modificações em dados sensíveis, execução arbitrária de código e, por fim, modificações em eventos de exclusão de ficheiros ou processos.

Pode gerar três tipos de relatório de resultados, variando em seu detalhamento de informações, e pode retornar em três formatos: texto simples, HTML ou objecto JSON, mas não categoriza de forma alguma a severidade ou precisão das vulnerabilidades encontradas.

2.3.7. EviCheck

Disponibilizada em 2015 pelos autores Mohamed Nassim Seghir e David Aspinall, esta ferramenta tem um objetivo diferente das mencionadas anteriormente: verificar, criar e certificar as políticas de segurança de uma aplicação Android, e trazer como resultado uma certificação digital da aplicação (Seghir & Aspinall, 2015).

Os requisitos para esta ferramenta são, uma aplicação APK e a política de uso permitida para esta aplicação, portanto deve-se a princípio criar uma política de uso. Esta política é, basicamente, as permissõe e negações para o uso legítimo da aplicação, ou seja, o que ela tem ou não permissão para fazer depois de instalada e executada. O objetivo para a criação de uma política de uso é incluir o máximo de métodos que são possivelmente maliciosos como não permitidos, e permitir o máximo de métodos que comumente são legítimos, e então, aplicar sobre um APK através do EviCheck.

Com uma aplicação APK e as políticas de uso definidas para este APK, é possível seguir por dois caminhos com o EviCheck: criar um certificado digital, onde o trabalho do EviCheck é de comparar os métodos negados e permitidos com os que o APK efetivamente faz e retornar um relatório das regras seguidas ou não cumpridas, ou criar um certificado digital de uma cerfiticação que já foi criada anteriormente pelo EviCheck, ou seja, validar uma certificação (Seghir & Aspinall, 2015), conforme mostra a *Figura 5*.

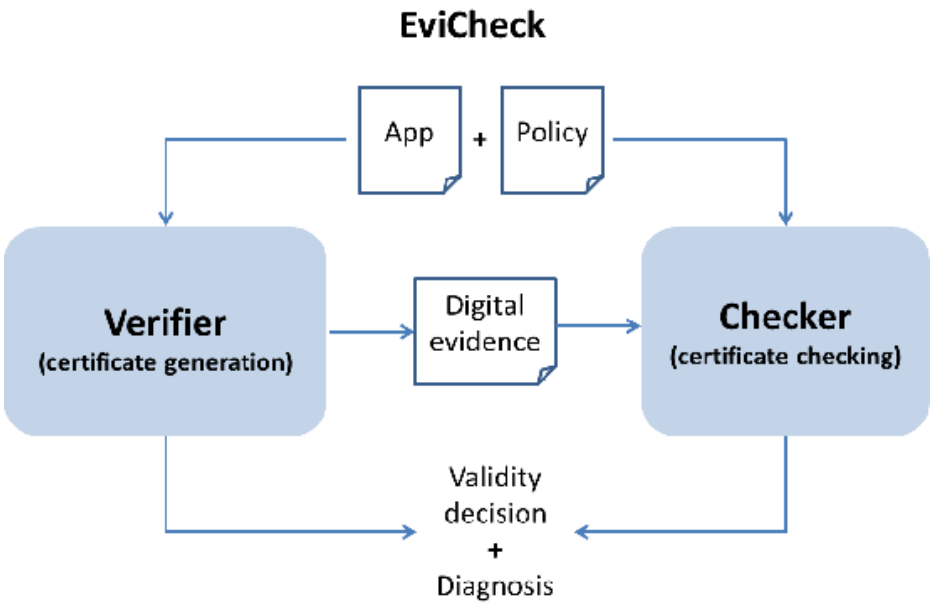

Figura 5. Estrutura da ferramenta EviCheck (Seghir & Aspinall, 2015).

Apesar de poderosa, esta ferramenta depende do utilizador para trazer resultados de vulnerabilidades, pois a política de uso é a base da verificação, e ainda, não categoriza a severidade do que encontra, em contrapartida a grande vantagem é a certificação digital que pode ser avaliada pelo próprio EviCheck.

2.3.8. QARK (Quick Android Review Kit)

LinkedIn é uma rede social de negócios fundada em 2003 (LinkedIn, 2019b), que entre muitos projetos, tem uma equipa para controle de ciber segurança denominada LinkedIn's House Security, em que, no ano de 2015, anunciou um projeto de código aberto para aprimorar a segurança de aplicações Android, e a chamou de Qark ou Quick Android Review Kit. Tony Trummer e Tushar Dalvi foram os criadores da idéia do Qark, uma ferramenta de análise estática de aplicações Android que deve reconhecer as principais vulnerabilidades de segurança e apresentar um relatório de fácil compreensão ao utilizador final (LinkedIn, 2019a).

Para além de permitir testes customizados, o Qark assume ter uma instalação, configuração e comandos mais simples que ferramentas do mesmo intuito, resultados com *links* aos programadores para aprenderem mais sobre a vulnerabilidade, e um histórico de análises.

Enquanto outras ferramentas de pesquisa de vulnerabilidades baseiam-se em uma forma de engenharia reversa do ficheiro APK (comumente Androguard), o Qark utiliza de várias destas para, de acordo com sua documentação oficial, maximizar a extração de informações da aplicação (LinkedIn, 2019a).

Apesar do relatório de resultado pode ser criado em objecto JSON ou em HTML para visualização do utilizador final, a forma de utilização não é tão simples e depende, assim como outras ferramentas de mesmo objetivo, de comandos sem nenhuma interface amigável, como mostra o exemplo na *Figura 6*.

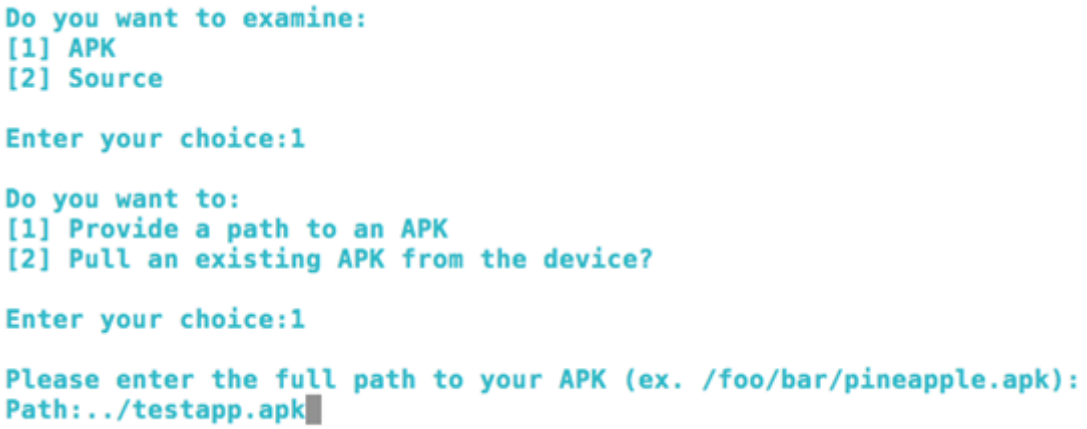

Figura 6. Exemplo de utilização do Qark.

2.3.9. StaCoAn

Com uma interface amigável ao utilizador, o StaCoAn se baseia na utilização de um servidor local e apresentação dos resultados de sua análise de vulnerabilidades sobre aplicações Android em HTML para que o utilizador possa navegar por entre as informações e analisar seus resultados. Vincent Cox é o criador do projeto que foi disponibilizado ao público em 2017, e recebeu diversas contribuições para a criação do StaCoAn, os programadores com contribuição mais significativa estão nomeados como autores no repositório oficial da ferramenta (Cox, 2018).

Esta ferramenta busca essencialmente pelas seguintes vulnerabilidades: credenciais que estejam referenciadas diretamente no código, chaves de entrada ou links a API que estejam diretamente no código, dados sensíveis não encriptados, e erros comuns na criação de código que não segue as boas práticas. E teve como objetivo a criação de uma

interface gráfica, como apresenta a *Figura 7*, para fácil visualização e navegação dos resultados, e diferente das outras ferramentas de mesma função, o StaCoAn não necessita de comandos e parâmetros para iniciar sua análise, pois disponibiliza uma opção *drag-and-drop* de ficheiros APK diretamente ao browser, ou seja, ao arrastar um ficheiro APK ao servidor montado localmente do StaCoAn, a análise inicia-se.

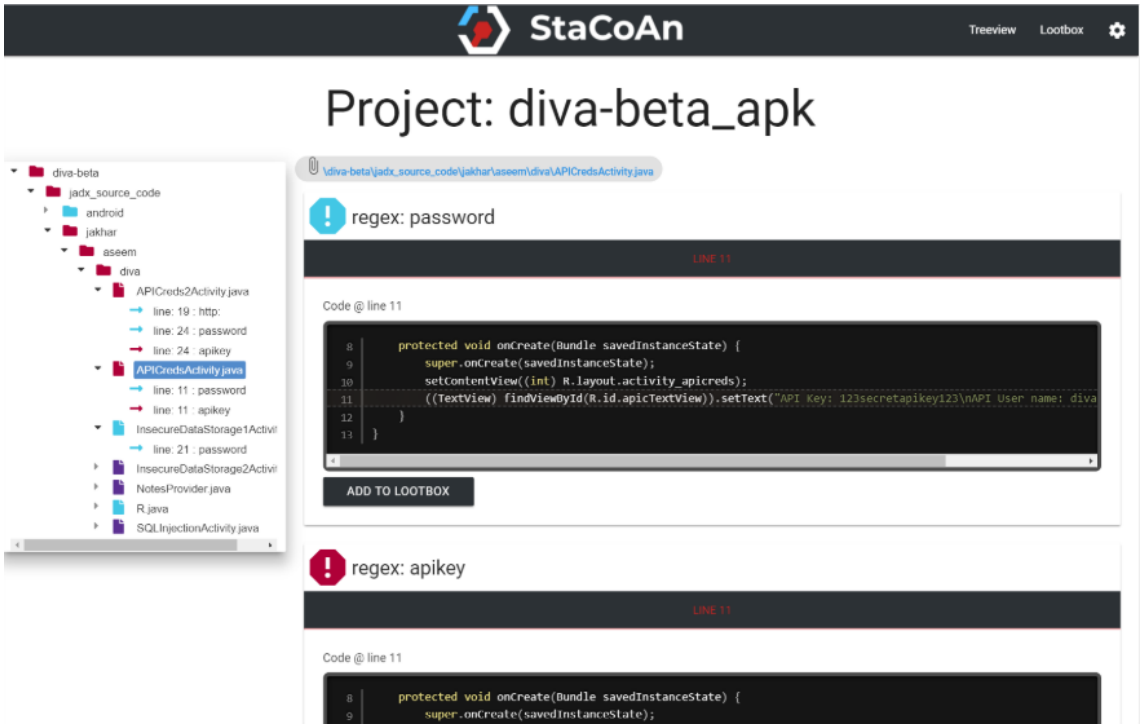

Figura 7. Interface gráfica de navegação dos resultados do StaCoAn (Cox, 2018)..

Por retornar apenas um resultado em formato visualmente agradável e não em um ficheiro JSON, ou qualquer outro formato, dificulta sua inclusão em um sistema de maior dimensão de detecção de vulnerabilidades..

2.4. Comparação das ferramentas

Todas as ferramentas apresentadas no subcapítulo anterior, têm o mesmo objetivo inicial de detectar vulnerabilidades em aplicações Android, mas com o foco diferenciado, algumas na forma de detecção, outras na melhor forma de apresentar os resultados. A

Tabela 4 se propõe em comparar as ferramentas baseado no problema apresentado por esta tese e inclusão no sistema DAAV, de acordo com as seguintes colunas:

- **Ferramenta**: Nome da ferramenta
- **Instalação**: Nível de dificuldade para instalar e configurar a ferramenta.
- **Análise**: Nível de dificuldade para iniciar o processo de análise de vulnerabilidades.
- **Qualidade**: Nível da qualidade de detecção de vulnerabilidades.
- **Relatório**: Nível de facilidade para o utilizador ler o resultado da ferramenta.
- **Categorização**: Existência ou não de uma categorização por severidade.
- **Feedback**: Existência ou não de um retorno de como resolver a vulnerabilidade detectada pela ferramenta.
- **DAAV**: Possibilidade de inclusão da ferramenta em um sistema maior como o proposto por esta tese.

Os níveis de dificuldade de utilização dos itens referenciados nas colunas da *Tabela 4* variam, do maior para o menor, de *complexo* para *médio*, *simples* e *inexistente*, assim como o nível de qualidade que tem a melhor pontuação como *alto*, em seguida *baixo* e por fim *inexistente*. A coluna de *categorização* tem um indicador diferente das outras, apresenta a completa inexistência de categoria nos resultados da análise da ferramenta, ou a utilização de uma categorização própria, que só é utilizada por essa ferramenta, e por fim, as que utilizam categorização da OWASP Top Ten.

Ferramenta	Instalação	Análise	Qualidade	Relatório	Categorização	Feedback	DAAV
Androguard	Complexa	Complexa	Alto	Baixo	Inexistente	Inexistente	Sim
ApkTool	Complexa	Inexistente	Inexistente	Inexistente	Inexistente	Inexistente	Sim
Androbugs	Complexa	Complexa	Alto	Baixo	Própria	Inexistente	Sim
Cuckoo S.	Complexa	Complexa	Alto	Médio	Inexistente	Inexistente	Sim
DroidStatX	Complexa	Complexa	Alto	Alto	OWASP	Sim	Sim

AndroWarn	Complexa	Complexa	Médio	Baixo	Inexistente	Inexistente	Sim
EviCheck	Complexa	Complexa	Baixo	Baixo	Inexistente	Inexistente	Sim
Qark	Média	Média	Alto	Alto	Própria	Sim	Sim
StaCoAn	Simples	Simples	Alto	Alto	Própria	Sim	Não

Tabela 4. Comparação das ferramentas de detecção de vulnerabilidades.

Apesar de praticamente todas as ferramentas descritas anterioremente terem a possibilidade de inclusão em um sistema maior como o DAAV, varia o nível de dificuldade para que essa integração, algumas necessitam de pequenas alterações em seu código, outras, que por exemplo, já retornam um resultado em formato JSON, não precisam de modificação alguma.

Ao seguir o raciocínio da *Tabela 4*, o sistema DAAV teria as características conforme a *Tabela 5.*

Ferramenta	**Instalação**	**Análise**	**Qualidade**	**Relatório**	**Categorização**	**Feedback**
DAAV	Inexistente	Simples	Máxima	Alto	OWASP	Sim

Tabela 5. Características do sistema DAAV de acordo com a Tabela 5.

DAAV não requer instalação, pois se propõe a estar disponível em um *website* de acesso livre por qualquer *browser*, de análise simples como StaConAn que necessita apenas escolher o ficheiro a ser analisado, de qualidade considerada máxima porque contem todas as ferramentas anteriores de análise, relatório de fácil percepção do utilizador categorizado pelo nível de severidade OWASP e ainda, não apenas apontar as vulnerabilidades mas apresentar um retorno ao programador de como resolver os problemas de segurança.

Capítulo 3 – Desenvolvimento

Este capítulo dedica-se ao desenvolvimento da ferramenta, designada por DAAV (*Detection of Android Aplication Vulnerabilities*), que apresenta a solução para o problema descrito no capítulo um, iniciando com a estrutura e arquitetura, diagrama de sequência, as linguagens de programação escolhidas, e explicação da codificação da ferramenta bem como da base de dados, uma passagem sobre os *plugins* de análise de vulnerabilidades e a normalização de resultados.

3.1. Introdução

DAAV é uma ferramenta que tem como objetivo analisar aplicações mobile de sistema Android e retornar suas vulnerabilidades e sugestões de correção. É resultado de uma pesquisa bibliográfica sobre o funcionamento das aplicações maliciosas e de um estudo sobre programação de um sistema baseado em código seguro, e as ferramentas que podem ajudar a encontrar as vulnerabilidades do código de uma aplicação.

Foram encontradas diversas ferramentas de análise de código Android e exposição de vulnerabilidades, cada uma delas com um tipo e formato diferente de resultado, estas respostas são analisadas, normalizadas e devolvidas ao utilizador, criando-se assim o DAAV.

3.2. Estrutura e Requisitos

O sistema pode ser dividido em cinco grandes componentes, conforme a *Figura 8*, que para facilitar a explicação durante este estudo, são denominadas: *server*, *manager*, *database*, *plugins* e *aplicação web*. Estes componentes esperam por uma entrada de dados, e armazenam ou fornecem dados a outro componente. Toda a estrutura deve receber um pacote Android (*Android Package* ou APK) e devolver uma análise de vulnerabilidades sobre este APK como resultado.

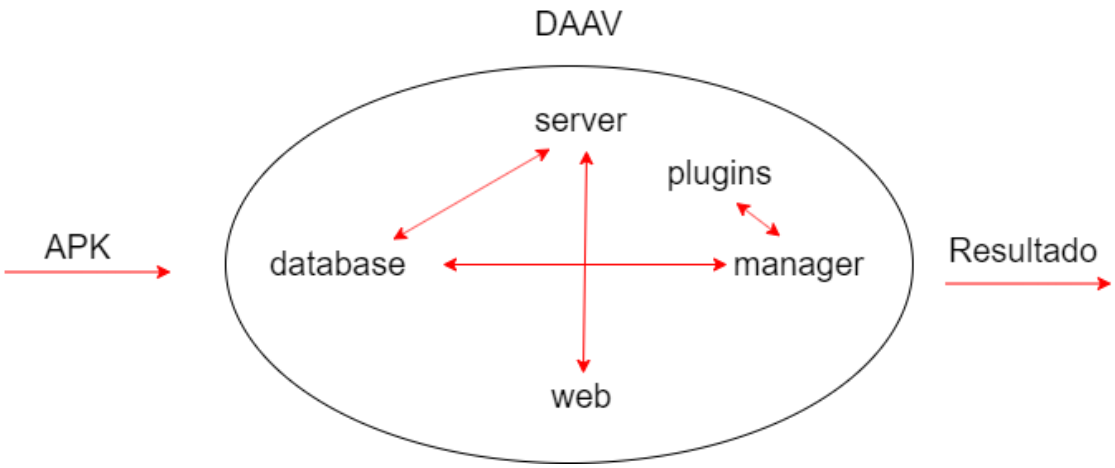

Figura 8. Estrutura do sistema DAAV

A *aplicação web* é a interface de comunicação com o utilizador, portanto, porta de entrada para as informações ao sistema, além do ambiente externo comunica-se apenas com o componente *server*, do qual recebe os resultados das análises. A principal informação externa que a *aplicação* web recebe é o pacote Android (APK) que será o material principal de análise, e pode ser enviado como um ficheiro APK ou, como característica específica do DAAV, um *hash* MD5 que identifica o APK na loja virtual Aptoide. Esta loja virtual foi escolhida pela vantagem de identificar todas as aplicações que tem disponíveis em seu acervo individualmente com *hash* único (Aptoide Team, 2019), e portanto, evita que o usuário tenha que efetuar o *download* do ficheiro APK localmente para análise, bastando para isso indicar o seu identificador MD da loja.

O componente *server* tem contacto com a *aplicação web* e *database*, tanto para envio como recebimento de informações, e está sempre ativo a espera de pedidos da *aplicação web* e basicamente serve como um filtro de informações entre os componentes que tem contacto. Recebe da *aplicação web* pedidos de análise de APK ou pedidos de visualização dos resultados das análises, e armazena na *database* APKs que devem ser analisados ou busca por resultados de análises neste mesmo componente.

Manager tem contacto com os *plugins* e com a *database* e corre independente do *server*, e não fica à espera de informações como tal, este é executado por um temporizador, e cada vez que inicia, busca na *database* por APKs que ainda não foram analisados, armazenados anteriormente pelo *server* a pedido da *aplicação web*. Envia os APKs encontrados na *database* para os *plugins* e pede pela inicialização de cada um deles, e recebe dos *plugins* os resultados das análises, que por fim, armazena na *database*.

Os *plugins* são ferramentas de análise de vulnerabilidades em ficheiros APK, recebem do *manager* os APKs a serem analisados, e retorna os resultados ao mesmo componente. O sistema DAAV pode conter diversos *plugins*, portanto a quantidade de análises devolvidas ao *manager* será a multiplicação do número de APKs enviados pela quantidade de *plugins* existentes.

A base de dados ou componente *database* é um repositório de informações que todo o sistema utilizará para armazenamento e troca de informações entre seus outros componentes, e tem contacto com o *server* e o *manager*. Recebe do *server* os APKs a serem armazenados para análise e devolve a este os resultados, enquanto envia ao *manager* os APKs e recebe deste os resultados.

3.3. Funcionalidade dos componentes

Cada componente do sistema DAAV tem funções específicas que dependem das informações que recebem ou do componente que estão a interagir, esta sessão descreverá estas funções separadas por componentes, de forma que na sessão seguinte seja possível apresentar linguagem de programação escolhida para cada um deles, e por fim a interação entre eles.

3.3.1. Componente *server*

Este componente contém pontos de acesso que ficam à espera de um pedido da *aplicação web* e funciona como uma API (Application Programming Interface) para a troca de informações, ou seja, como um contrato feito entre o servidor e o cliente, onde o servidor entrega os resultados de acordo com o pedido do cliente. A *Tabela 6* apresenta todos os pontos de acesso do *server*, os dados que recebe e o retorno em caso de sucesso ou falha, nota-se ainda uma subdivisão destes pontos de acesso, três envolvem manipulação das informações do utilizador, e outros três os dados que envolvem a análise de vulnerabilidades da aplicação móvel.

Ponto de acesso	Dados de entrada	Sucesso	Falha
register	*username, password*	Mensagem de sucesso	Mensagem de erro
login	*username, password*	Informações do utilizador e token JWT de acesso	Mensagem de erro
edit_profile	Dados do utilizador	Mensagem de sucesso	Mensagem de erro
apkscan	Ficheiro APK, *hash* MD5	Mensagem de sucesso	Mensagem de erro
apkfeedback	Identificação do APK	Resultado das análises de um ficheiro APK	Mensagem de erro
resultlist	*username*	Lista dos resultados de análise dos ficheiros APK enviados por este utilizador	Mensagem de erro ou lista vazia

Tabela 6. Pontos de acesso do componente server.

A manipulação de dados do utilizador se faz através de três pontos no servidor denominados *register, login* e *edit_profile*, e os que envolvem os dados de análise de vulnerabilidades são *apkscan*, *apkfeedback* e *resultlist*.

O ponto de acesso *register* tem função registar um novo utilizador para ter acesso a ferramenta, tem como dados de entrada o nome do utilizador (*username*) que deve ser único e uma senha (*password*) que será encriptada antes de ser armazenada. Ao ser invocado, este ponto de acesso tentará armazenar os dados enviados em uma tabela da base de dados, e caso encontre o nome de utilizador já existente na tabela, ou qualquer

falha de comunicação, retorna uma mensagem de erro, e no caso positivo armazena os dados e devolve uma mensagem de sucesso.

Assim como *register*, o ponto de acesso *login* recebe nome de utilizador e senha, mas ao invés de tentativa de registo na base de dados, este compara com os dados armazenados pela função *register* e retorna erro caso não encontre dados idênticos. No caso positivo o retorno é um token de acesso JWT (JSON Web Token), com duração de 60 minutos, que confirma a ligação de segurança com a *aplicação web,* acrescido dos dados do utilizador que são preenchidos em *edit_profile*, e caso ainda não as tenha, serão devolvidos campos por defeito a espera de edição.

A edição dos dados pessoais do utilizador é feita pela *aplicação web* e enviado ao ponto de acesso *edit_profile*, que em resposta positiva, substitui na base de dados os campos de dados pessoais anteriores, para que tenha este retorno positivo o token JWT é verificado em sua autenticidade e prazo de validade, em caso de recusa nenhum dado é alterado e o utilizador perde acesso a ferramenta.

O ponto de acesso *apkscan* recebe um ficheiro APK e tenta armazena-lo no componente *database*, caso não encontre nenhum ficheiro idêntico, retorna a mensagem de sucesso e efetivamente o armazena, caso contrário recebe uma mensagem de erro. Este ponto de acesso pode ainda receber um MD5, nesta situação o *server* efetuará o *download* do ficheiro APK na loja Aptoide e seguirá com os passos como se recebido um ficheiro APK.

A *aplicação web* pede pela lista de resultados de análises dos APKs pelo ponto de acesso *resultlist*, e envia para este ponto o *username,* para que o *server* possa enviar como resposta apenas os resultados de análise deste utilizador. O retorno positivo consiste na lista de APKs analisados que foram enviados por este utilizador, e procede dessa forma quando existem resultados registados na *database* vinculados ao *username* enviado pela *aplicação web*, caso contrário a resposta é uma lista vazia, ou em caso de falha, como falta de conexão, o retorno é uma mensagem de erro.

O *apkfeedback* é o ponto de acesso que é acedido pela *aplicação web* ao escolher os detalhes da análise de resultados de um APK em específico, e para tanto, precisa receber a informação de qual APK deve procurar. Caso encontre o resultado de análise, o retorna ao utilizador, caso contrário uma mensagem de erro.

3.3.2. Componente *manager*

O componente *manager* é executado por um temporizador, e a cada execução, busca no componente *database* por APKs armazenados pelo *server* que ainda não tenham sido analisados. Caso positivo, procura no componente *plugin* por ferramentas de análise de vulnerabilidades, e cria uma lista de *plugins* disponíveis. Em seguida inicia a execução de todos estes *plugins* em paralelo sobre cada APK encontrado, ao fim de cada análise de *plugin* o componente *manager* armazena o resultado da análise, nome do *plugin* e do utilizador, assim como dados do APK, no componente *database*.

O *manager* tem ainda uma função de extrema importância para o sistema DAAV, ao fim de todas as análises dos *plugins* sobre um APK, executa uma subrotina que normaliza as diferentes respostas, e as categoriza de acordo com a metodologia OWASP Mobile Top Ten, e então armazena este resultado no componente *database*. Todos os resultados gerados pelo *manager*, de *plugins* indepentendes ou da união de todos e categorização, são armazenados em formato JSON para que possam ser trabalhados na *aplicação web*.

3.3.3. Componente *plugins*

Este componente é composto efetivamente das ferramentas de análise de vulnerabilidades exemplificadas no ítem **2.3** deste trabalho. Tem início a pedido do componente *manager*, que lhes envia um ficheiro APK para análise e trabalham individualmente sobre este ficheiro, resultando em objectos JSON que contém as repostas das análises de cada *plugin*.

Cada *plugin* trabalha da forma com que foi criado, mas sofreram pequenas alterações para que pudessem estar integrados ao sistema DAAV, basicamente na forma de entrada e saída de dados de cada um. A entrada de dados deve aceitar o APK enviado pelo componente *manager* e iniciar automaticamente sua análise, e a saída foi padronizada para retornar sempre um objecto JSON para que o componente *manager* possa categorizar todas as respostas e armazenar no componente *database*.

3.3.4. Componente *database*

A base de dados ou componente *database* tem a função de armazenar todas as informações geradas pelos demais componentes, e serve como acervo de pesquisa para os mesmos.

Armazena os APKs a serem analisados a pedido do *server* e valida que o APK adicionado é único, portanto, não gera incoerência de dados. Guarda também os resultados gerados pelos *plugins* e categorizados pelo *manager* para que possam ser acedidos a pedido da *aplicação web* pelo *server*, e ainda, mantém um acervo de informações de APKs já analisados para que não seja feita uma análise duplicada. E por fim, armazena os dados de utilizador, para verificação de acesso ao sistema DAAV e retorno de informações pessoais.

3.3.5. Aplicação web

A *aplicação web* tem como principal objetivo entrada e saída de dados para o utilizador final do sistema DAAV, recebe todos os dados externos como informações do utilizador e APK para análise, e se comunica apenas com o componente *server*. Para que se inicie qualquer atividade no DAAV, o utilizador deve-se registar e em seguida fazer *login*, como exemplificado na *Figura 9*, dessa forma qualquer atividade do utilizador ficará registada em seu *username* que é único no componente *database*. Este sistema possibilita buscar informações relevantes apenas a este utilizador, gerar estatísticas das análises feitas por este utilizador, e criar histórico de análises.

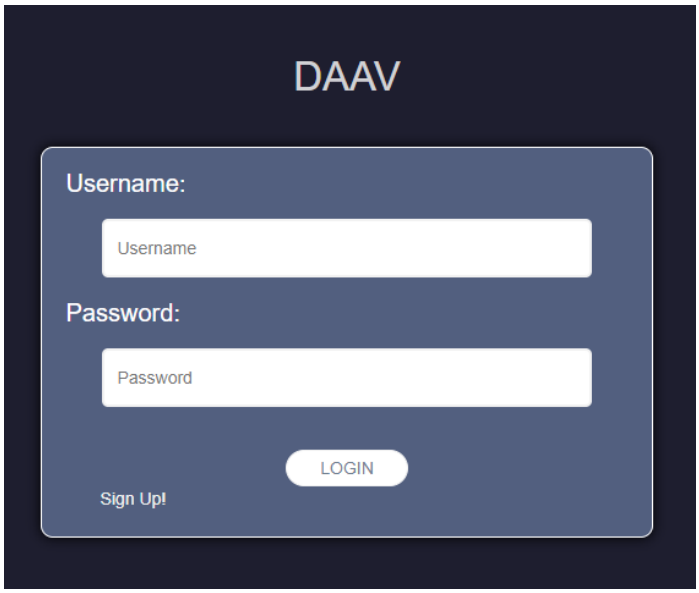

Figura 9. Registo e login de utilizadores no sistema DAAV.

Após login o utilizador tem acesso a cinco páginas da *aplicação web:* estatística de análises, envio de APK, lista de resultados, detalhes do resultado e edição de perfil. As estatísticas estão na página principal e, a princípio, contém três gráficos que apresentam do total de análises feitas, o número de vulnerabilidades encontradas categorizadas pela OWASP, o número de vulnerabilidades da última aplicação analisada, e o total de análises feitas pelos meses. Esta página, exemplificada na *Figura* 10, é dinâmica e a cada análise feita, acresce o número estatístico dos gráficos descritos, e ainda, está criada de tal forma que a inclusão de novos gráficos é feita de maneira simples com base nos dados do componente *database*.

Figura 10. Página de análise estatística do DAAV.

Com acesso através do ícone no canto superior direito, o utilizador pode editar seus dados de perfil, com exceção de seu *username* que foi escolhido ao registar-se no início da aplicação, e é a chave de identificação deste utilizador. As informações desta página são apenas para visualização do próprio utilizador e, até então, não são utilizadas para a funcionalidade do sistema DAAV, e é apresentado na *Figura 11*.

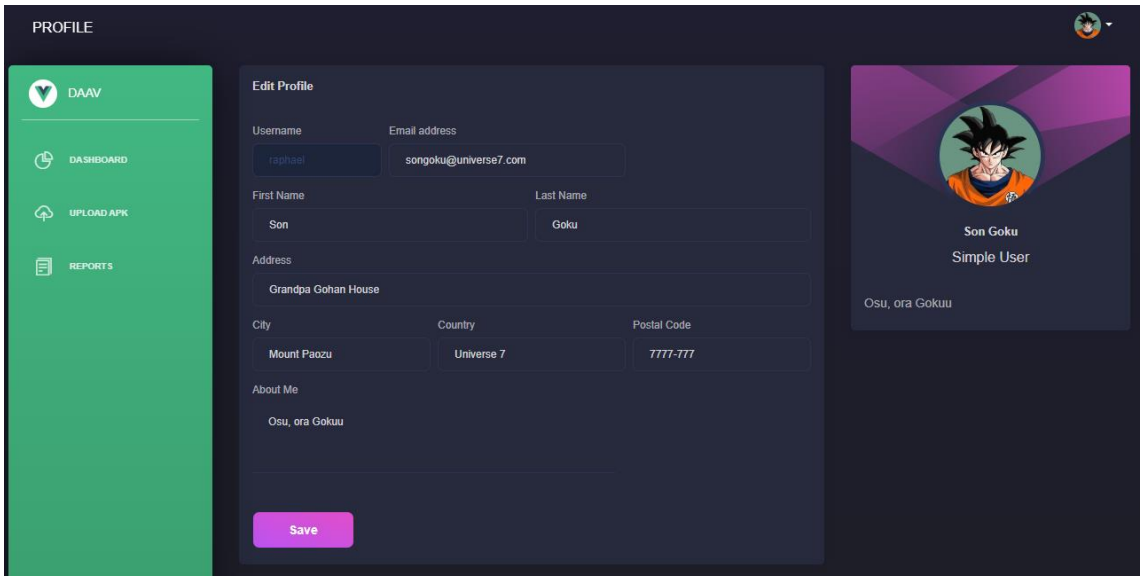

Figura 11. Página de dados do perfil do utilizador do sistema DAAV.

Assim como a primeira página de dados estatísticos, a página de envio de APK pode ser acessada pelo menu lateral esquerdo, e possibilita o envio de um ficheiro APK que está armazenado localmente no computador do utilizador, conforme *Figura 12*, ou a escolha de um MD5 da uma loja virtual Aptoide, que identifica um APK (Aptoide Team, 2019), e disponibiliza esta informação em seu *website* ao escolher visualizar os detalhes de uma aplicação.

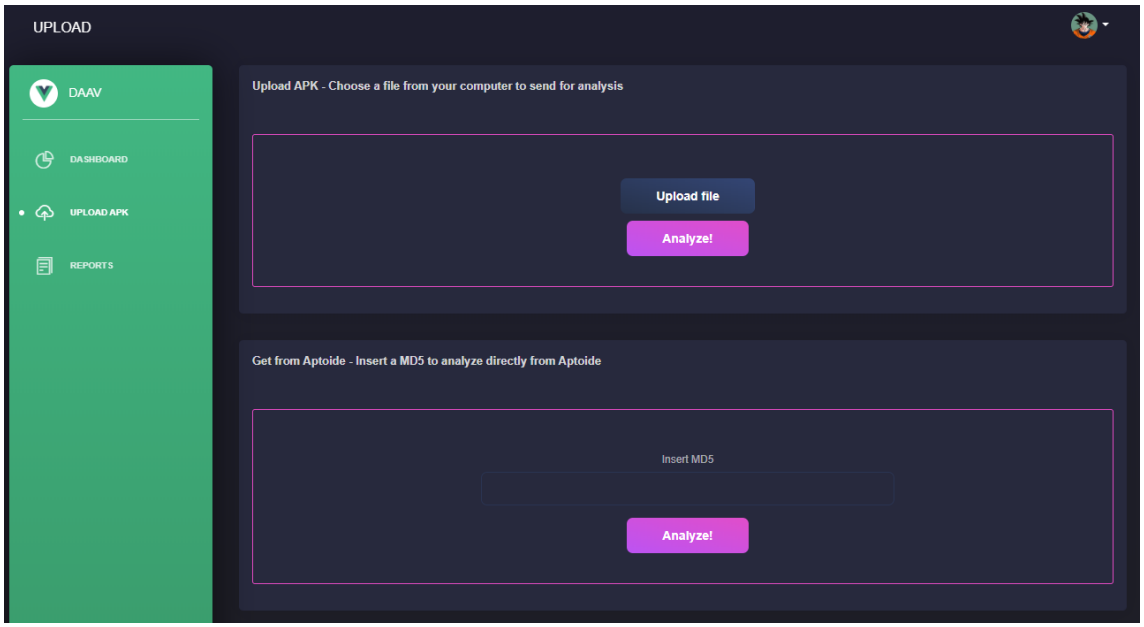

Figura 12. Página de envio de apk para análise do sistema DAAV.

Após envio de aplicações para análise, os outros componentes do sistema DAAV serão ativados e disponibilizarão o resultado para a *aplicação web* buscar e apresentar na página que contém a lista de resultados, conforme *Figura 13*. As colunas da lista de resultados contêm o nome do APK analisado, seu hash MD5, o *plugin* que gerou o resultado ou a forma categorizada contendo a união dos resultados de todos os *plugins*, a data de criação da análise e um *link* para detalhes, que devolverá ao utilizador um relatório detalhado. Os detalhes das análises individuais de cada *plugin* é um retorno do objecto JSON de cada um, com suas informações características, exemplificado na *Figura 14*, mas se a linha não conter o nome do *plugin* e sim a informação "*Categorized!",* o *link* de detalhe levará ao relatório normalizado de todos os *plugins* anteriores e categorizado com a metodologia OWASP Mobile Top Ten, e ainda, acrescidos com links de *feedback* para que o utilizador

não apenas vejas as vulnerabilidades, mas saiba como resolvê-las, conforme exemplificado na *Figura 15*.

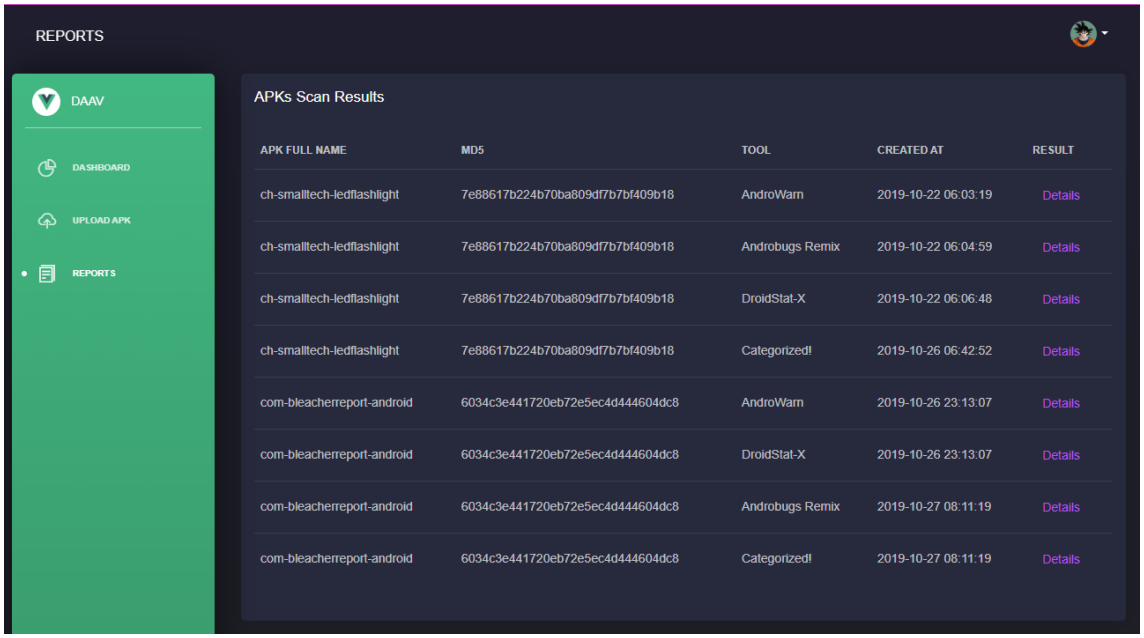

Figura 13. Lista de resultados de análises do DAAV.

A página de detalhes das análises consiste no retorno do *server* de acordo com a opção escolhida na *aplicação web.* A escolha de um *link*, que é identificado em uma linha da lista de resultados e contenha o nome de um *plugin*, não utiliza a função de categorização do componente *manager* e envia apenas o resultado deste *plugin* individualmente, conforme *Figura 14.*

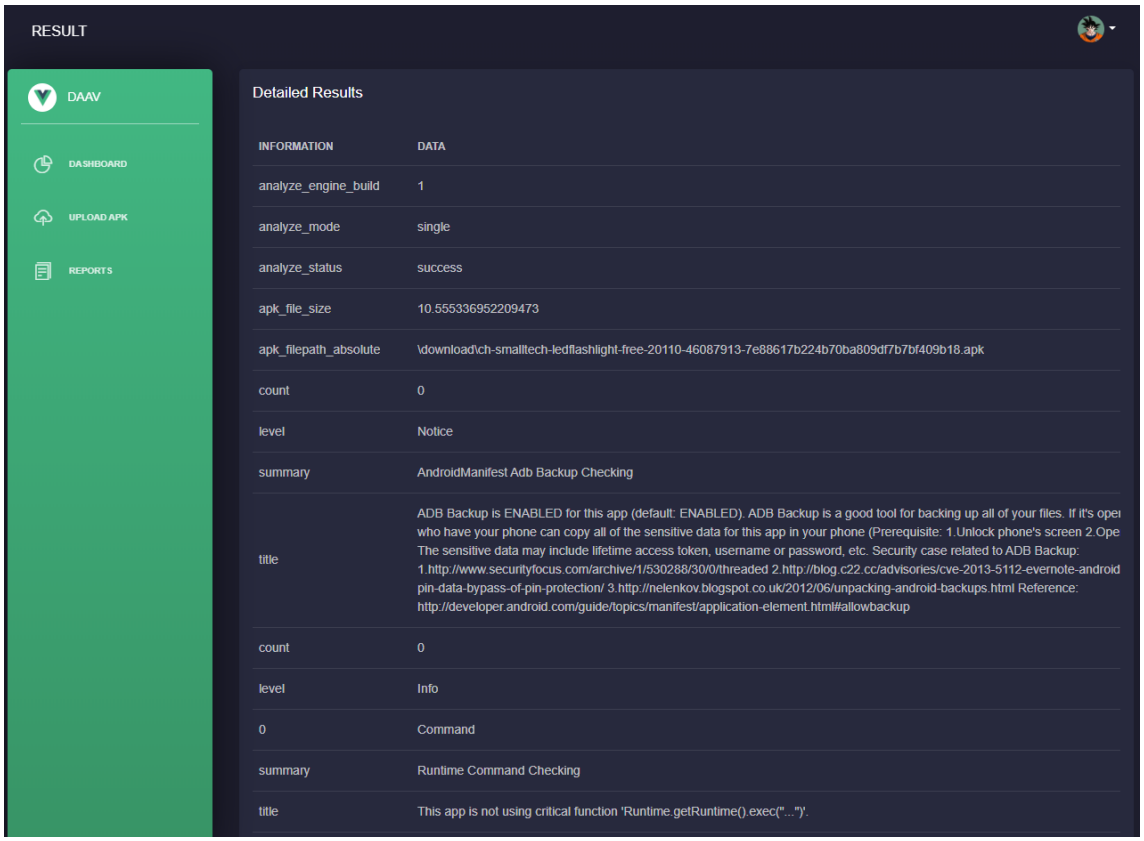

Figura 14. Exemplo de uma análise individual de um plugin no sistema DAAV.

O retorno apresentado na *Figura 14* não acrescenta nada novo a resposta do *plugin* que já existe, exceto pelo fato de que é apresentado dentro do sistema DAAV e, portanto, tem acesso muito mais simples do que o uso da ferramenta por si só, a excluir todo o processo de instalação e configuração da ferramenta em questão, e ainda, armazenamento do resultado para análise futura. Os campos apresentados neste tipo de detalhamento são dinâmicos e variam de acordo com a resposta do *plugin*.

A *Figura 15* apresenta o resultado de todos os *plugins* inclusos no sistema DAAV, normalizados e categorizados pelo componente *manager*, acrescidos de *feedback* ao utilizador.

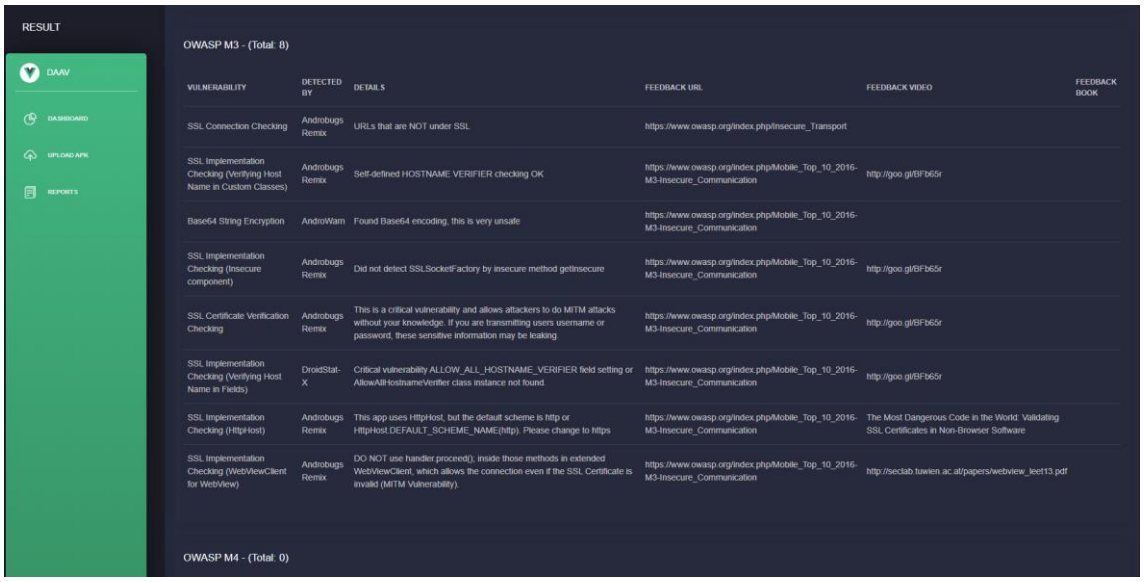

Figura 15. Exemplo de uma análise categorizada pelo sistema DAAV.

Neste caso, o detalhamento é dividido por tabelas pela metodologia OWASP Mobile Top Ten, que identifica o nível de severidade e contém a quantidade total de vulnberabilidades encontradas em cada categoria. As tabelas são divididas pelos campos *vulnerability*, *detected by*, *details* e *feedbacks* descritos em detalhes na *Tabela 7*.

Vulnerability	Todos os *plugins* identificam as vulnerabilidades encontradas com um título, estes são analisados e armazenado de forma a não gerar redundâncias.
Detected By	As vulnerabilidades são encontradas por um ou mais *plugins*, neste campo é exposto o *plugin* de maior relevância ao detectar a vulnerabilidade em questão.
Details	Uma detalhamento da vulnerabilidade detectada, de acordo com a metodologia OWASP .
Feedback URL	Apresenta um ou mais *links* com um maior detalhamento da vulnerabilidade detectada e indicações de como solucionar essa falha de segurança

Feedback Video	Apresenta um ou mais vídeos com um maior detalhamento da vulnerabilidade detectada e indicações de como solucionar essa falha de segurança
Feedback Book	Indica livros que contém mais informações e possíveis soluções a vulnerabilidade detectada.

Tabela 7. Estrutura de relatório de resposta as análises do DAAV.

O detalhe do resultado das análises de uma aplicação propõe-se em ser de fácil leitura ao utilizador, com credibilidade por utilizar de diversas ferramentas de análise sobre a mesma aplicação, e com a constante inclusão de *feedback*, não apenas com *links*, mas também com vídeos, livros ou artigos relacionados a vulnerabilidade.

3.4. Linguagem de programação

Para a criação desta ferramenta, foi realizado um estudo para escolha de linguagens de programação a serem utilizadas, e como a proposta não é de criação de uma ferramenta de análise, mas sim de normalização da resposta de inúmeras aplicações já existentes de teste de vulnerabilidades, a primeira linguagem a ser utilizada foi *python*, por existirem diversas ferramentas de análise de qualidade nesta vertende de programação.

Python é uma linguagem criada em 1991 por Guido van Rossum, um programador holandês que buscava, entre outros objetivos, por simplicidade de código, o que torna *python* uma linguagem que necessita de menos linhas de código para trazer os resultados esperados, o que não a torna menos poderosa (Foundation, 2016)

Os componentes *server*, *plugins* e *manager* foram criados em *python*, e por ter divergência de código entre versões desta linguagem, com excessão do *server* estes componentes foram criados em duas versões (*python* 3.6 e *python* 2.7).

Apesar da escolha de *plugins* ter sido feita baseada na linguagem *python*, alguns destes estavam escritos na versão 2.7 e outros na versão 3 ou superior, e como o código dos *plugins* foi alterado minimamente para integração com o *manager*, não era o objetivo re-escrever a ferramenta de análise em uma nova versão, portanto foram criadas chamadas

a *plugins* nas duas versões, uma para contemplar os plugins anteriores e outra os posteriores a versão 3 (Rossum & Drake, 2012).

Para a *aplicação web* foi escolhido a apresentação online via *browser* e, portanto, utilizado HTML e CSS como base de criação, mas estas duas linguagens não são suficientemente poderosas para apresentar toda a proposta desta ferramenta, que utiliza também de *javascript* e da *framework* Vue.js que completam o componente.

Vue.js é uma framework criada por Evan You em 2014 que tem êxito em manipular o DOM (Document Object Model ou Modelo de Objecto de Documentos) de forma simples e eficaz, ao unir variáveis de programação a apresentação no HTML e transformação dinâmica da visualização do utilizador final (Cromwell, 2016). Com essa eficácia de manipulação do DOM e, por ser *javascript*, a nativa manipulação de objectos JSON (JavaScript Object Notation) tornou-se ideal para apresentar os dados recebidos pelo *server* no *browser* para o utilizador.

JSON por sua vez, é um formato de notação para troca de dados, gerado por código, mas de fácil leitura humana e conversão para objecto de manipulação em todas as linguagens escolhidas para a ferramenta aqui proposta (Douglas, 2019), e é utilizada em todas as trocas de dados entre componentes do DAAV.

Para armazenamento e resgate de todas as informações deste sistema, foi escolhido o formato de base de dados relacional, por ser de rápido e fácil acesso as informações de um sistema do porte do DAAV (Wolfgang Keller, 2019), e MySQL é um sistema de gerenciamento de código aberto para esse tipo de base de dados (Pachev & Pachev, 2007).

A *Figura 16* acresce as linguagens de programação escolhidas a *Figura 8* apresentada no ítem **3.2**.

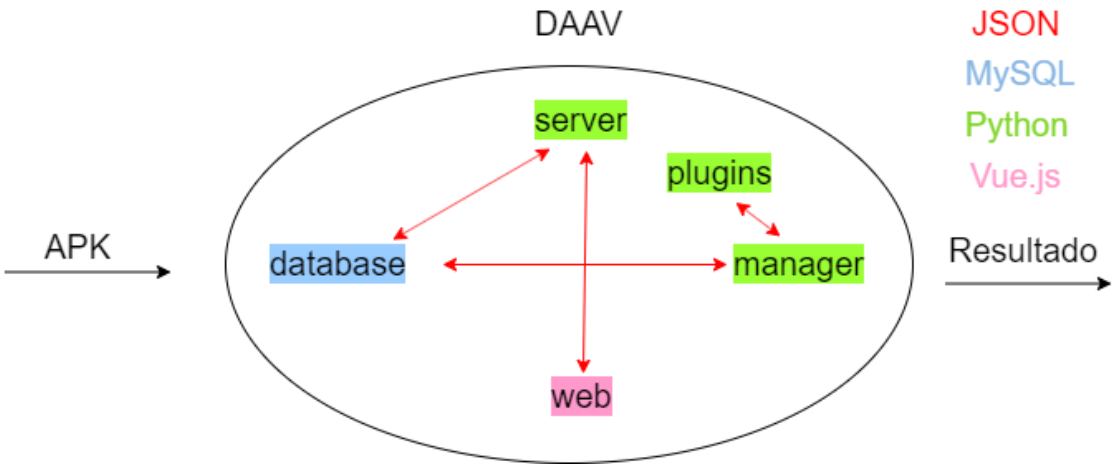

Figura 16. Estrutura do DAAV com separação de linguagem.

Para gerenciamento do componente *database* em MySQL, foram criadas quatro tabelas, para gerir utilizadores, APKs analisados, APK que estão à espera de serem analisados, e por fim de resultados das análises, conforme *Figura 17*.

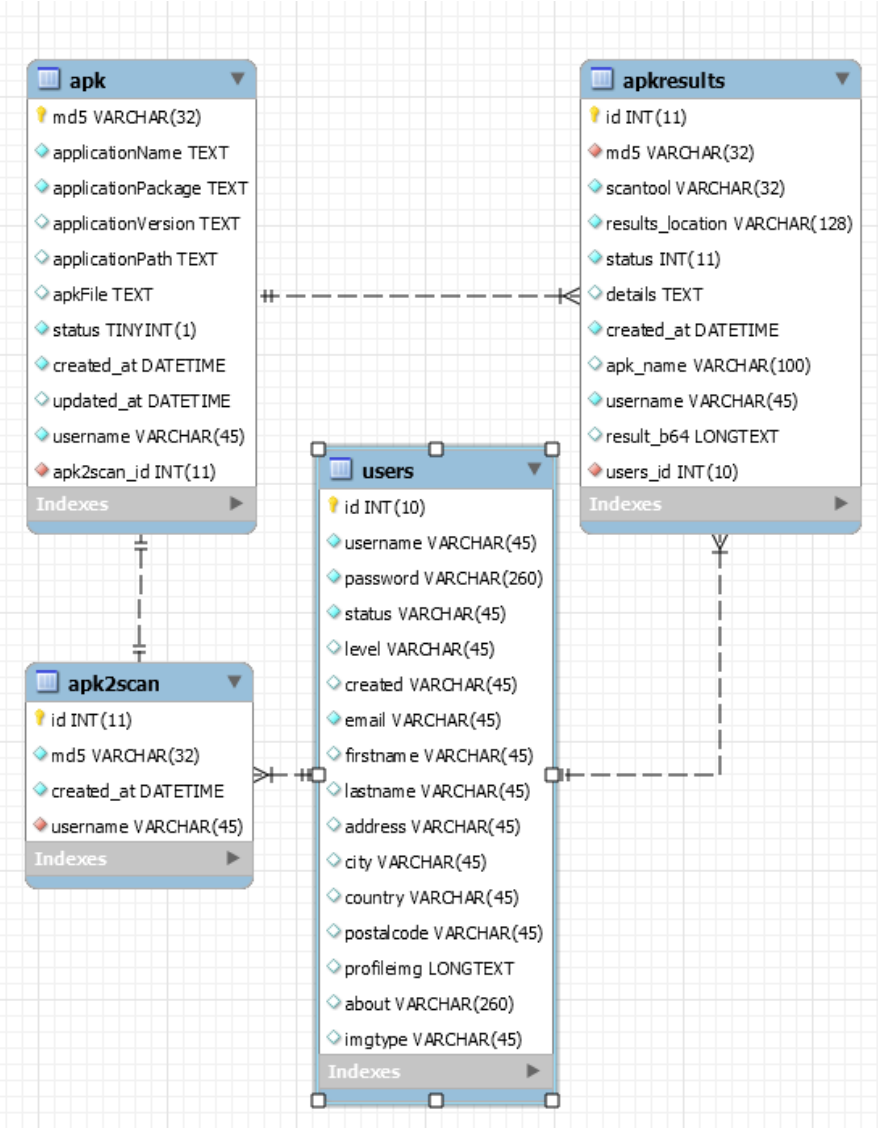

Figura 17. Tabelas do componente database do sistema DAAV.

A tabela de utilizadores, denominada *users* contém um nome de utilizador único e sua senha encriptada, bem como todos os dados que podem ser editados em sua página de perfil disponibilizada pela *aplicação web*, a tabela *apk2san* contém a identificação do utilizador e o APK submetido para análise, mas que ainda não foram analisados, a tabela *apk* contém informações dos APKs já analisados e o utilizador que o submeteu, e por fim a tabela *apkresults* contém os resultados individuais e os categorizados de cada APK e a ferramenta que o plugin executou.

3.5. Interação entre os componentes

Os componentes do sistema DAAV interagem entre si conforme a *Figura 16*, e nesta sessão terão um detalhamento do comportamento de cada interação, desde a entrada de dados do utilizador pela *aplicação web*, até a visualização do resultado categorizado em detalhes.

O contacto do utilizador com o sistema DAAV é feito pela *aplicação web*, e a primeira interação, pede pela entrada do nome do utilizador e uma senha, não somente para a permissão de utilização do sistema, mas também para identificação do utilizador, dos APKs que este enviará para avaliação e estatística de resultados. Os dados inseridos pelo utilizador através da *aplicação web* são enviados para o *server*, que no caso de registo deve verificar a existência de um *username* único e armazenar as informações do utilizador na *database*. Em caso de tentativa de *login*, faz uma busca na *database* pelo *username* e verifica a *password* encriptada, se o resultado for positivo envia um *token* que, enquanto estiver ativo, permite a navegação em outras páginas do DAAV, é enviado no mesmo pedido as informações pessoais do utilizador, caso existam, conforme fluxograma da *Figura 18*.

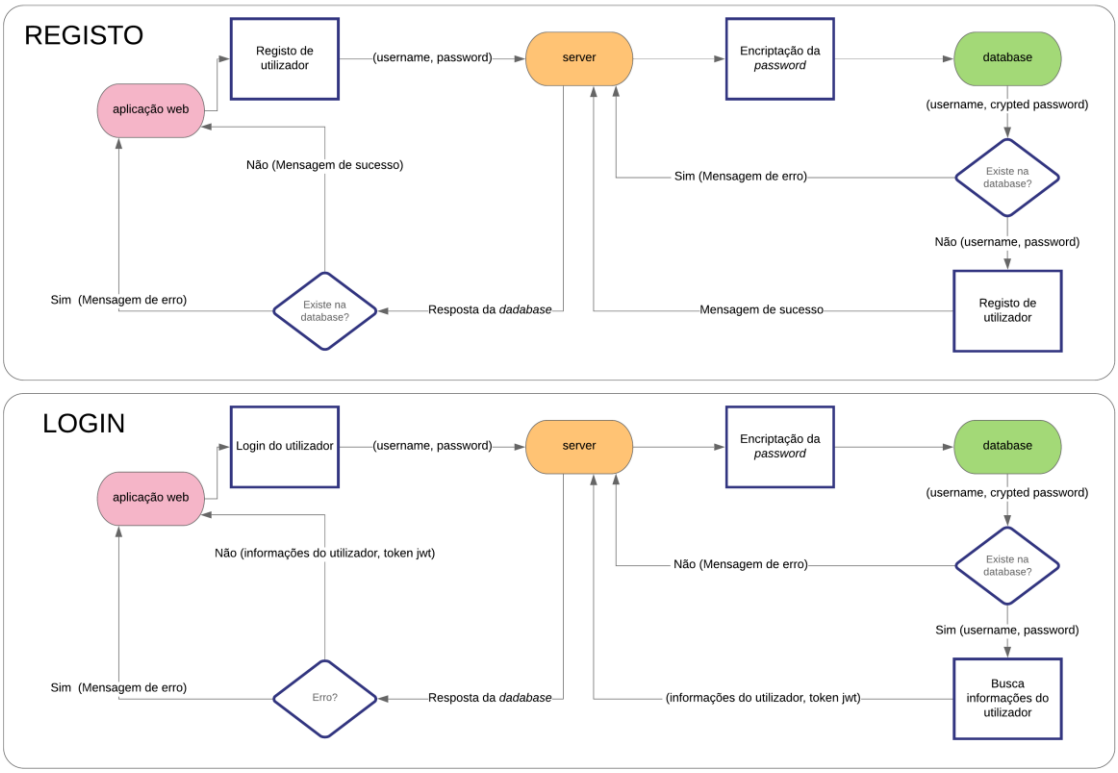

Figura 18. Fluxograma de registo e login do sistema DAAV.

Para além do sistema de registo e *login* do utilizador no sistema DAAV, a *aplicação web* e o *server* interagem com a transferência de APKs e resultados, o utilizador envia ao *server* o ficheiro ou MD5 que representa o APK ao *server*, que armazena na *database* em uma tabela de APKs a serem analisados, e ao pedir pelos resultados o *server* busca-os na *dabase* e retorna todos os referentes ao utilizador logado, como na *Figura 19*.

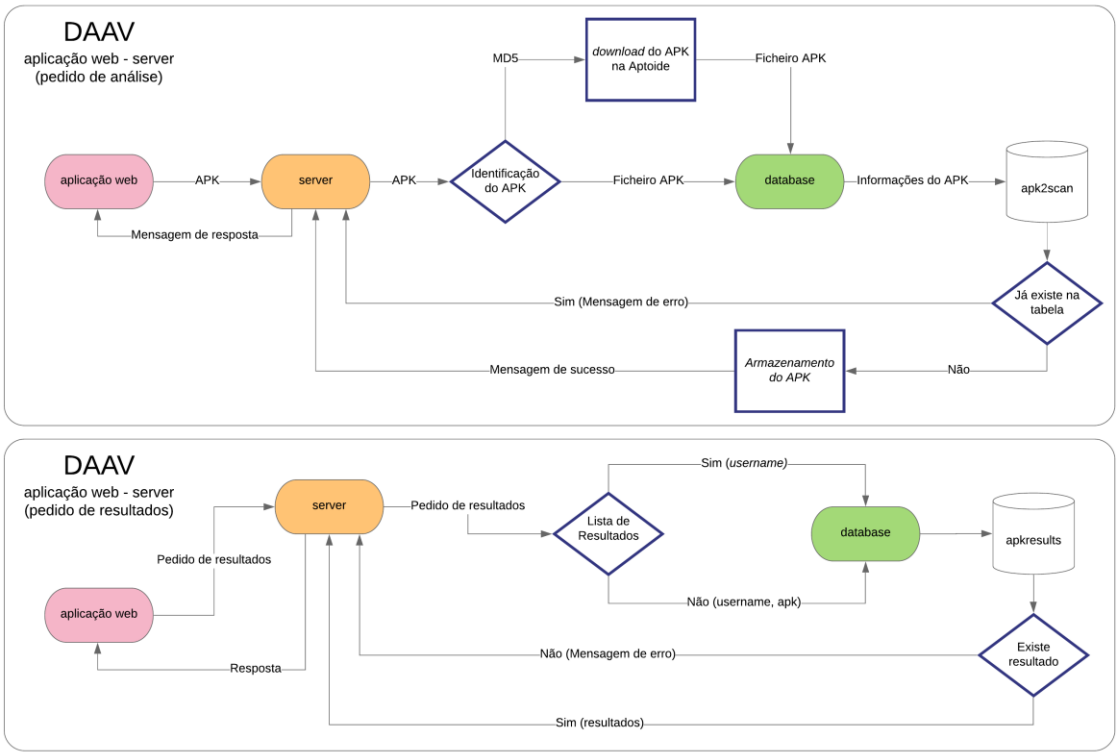

Figura 19. Fluxograma de pedidos referentes a APK e resultados da aplicação web e server.

O componente *manager* interage com os *plugins* e *database* e une as funções relacionadas a análise e categorização de vulnerabilidades com os pedidos da *aplicação web* e *server* ao armazenar na os resultados no componente *database*. Executado de forma indepentende por um temporizador, o componente *manager* busca os APKs a espera de análise na tabela *apk2scan* na *database* e os envia para os *plugins* efectuarem as análises. São executadas duas subrotinas que ativam os *plugins* em separado de acordo com sua versão: *python* 2 ou 3, estas trabalham exatamente da mesma forma, a separação existe apenas pela incoerência de código de cada uma das versões, e o DAAV suporta a execução de ambas.

A cada retorno de um *plugin*, o *manager* armazena a resposta na tabela *apkresults* da *database*, incluindo nesta resposta o nome da ferramenta de análise, qual foi o APK analisado e o utilizador que a pediu. Ao fim de todos os *plugins* executarem suas análises, o *manager* normaliza e categoriza os resultados baseado na metodologia OWASP Mobile Top Ten, e cria mais um relatório a ser armazenado na mesma tabela. Portanto o número de respostas que o *manager* armazenará na *database* será a quantidade de APKs multiplicado pela quantidade de *plugins*, acrescido de um. O fluxograma da *Figura 20* mostra a interação do *manager*, *database* e *plugins*.

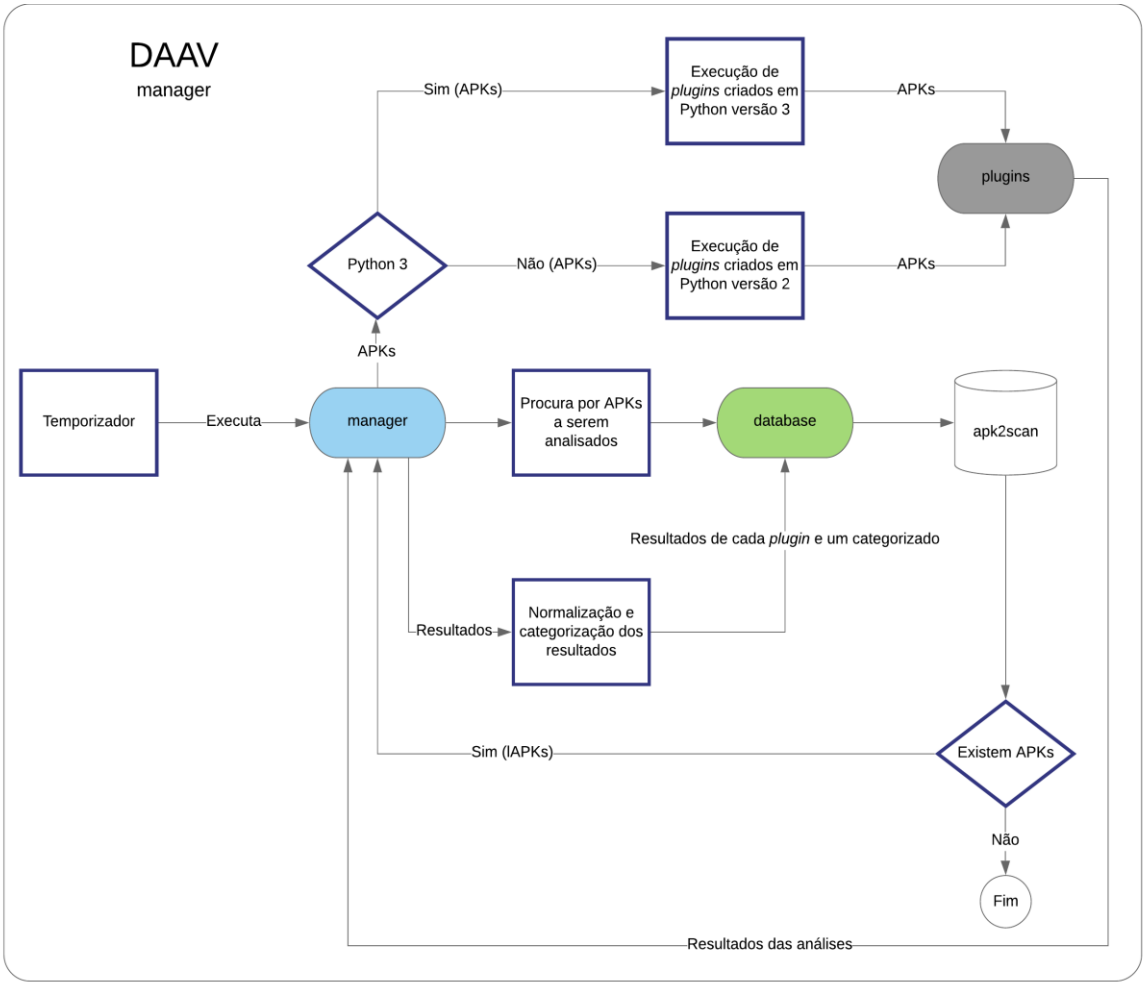

Figura 20. Fluxograma do componente manager no sistema DAAV.

Da forma com que está construído, o DAAV é escalável e pode receber inúmeros *plugins*, desde que recebam alterações para se adequarem ao modelo do sistema. Como cada *plugin* é independente, para se enquadrarem ao sistema é preciso normalizar a entrada e saída de dados do *plugin*, a solução encontrada foi a passagem de parâmetros de entrada pelo *manager*, e saída de dados do *plugin* ser toda em formato JSON.

Capítulo 4 – Análise e discussão dos resultados

Este trabalho propõe-se apresentar o protótipo de uma ferramenta *web* que analisa aplicações de dispositivos móveis em sistemas Android, e retornar ao utilizador, não apenas um relatório com as falhas de segurança encontradas, mas também uma normalização e categorização conforme a metodologia OWASP Mobile Top Ten, acrescido de *feedback* com indicações de como solucionar estas falhas, dessa forma, está a ajudar a criar aplicações mais seguras.

Durante o estudo de caso apresentado neste capítulo, a ferramenta DAAV está configurada para usar três *plugins* de análise de vulnerabilidades, e foram analisadas 20 aplicações Android, consideradas em 2019 como mais populares (na loja de aplicações Aptoide), como mostra a *Tabela 8*, dentre elas jogos, redes sociais, vídeo *streaming*, aplicações financeiras, conversação, mapas, compartilhamento de viagens, aplicações relacionadas com alimentação, música e fotos.

Nome	**Tipo**	**Número de Downloads (Aproximadamente em milhões)**	**Última atualização**
Clash Royale	Jogo	250	20/10/2019
Facebook	Rede Social	250	30/10/2019
Instagram	Rede Social	250	31/10/2019
Messenger	Conversação	250	30/10/2019
Polarr Photo Editor	Fotos	250	30/10/2019
Torch Flashlight LED HD	Lanterna	250	08/06/2019
WhatsApp Messenger	Conversação	250	29/10/2019
Clash of Clans	Jogo	50	28/01/2019

Pokemon Go	Jogo	50	24/10/2019
Snapshat	Rede Social	50	30/10/2019
Wish	Compras	50	31/10/2019
Dropbox	Núvem	25	29/10/2019
Hangouts	Conversação	25	23/10/2019
Netflix	Video *Streaming*	25	08/05/2019
Sixt Car Rent	Aluguel de veículos	25	30/10/2019
Spotify	Músicas	25	30/10/2019
VLC for Android	Videos	25	30/10/2019
Waze	Mapas	25	23/10/2019
Uber	Viagens	5	29/10/2019
LinkedIn	Rede Social	3	05/04/2019

Tabela 8. Aplicações inseridas no DAAV como estudo de caso.

As informações da *Tabela 8* foram retiradas do *website* da loja virtual Aptoide (Aptoide Team, 2019), e está filtrada do maior para o menor número aproximado de *downloads*, em milhões, e em seguida ordenada alfabeticamente pelo nome da aplicação.

As vinte aplicações foram submetidas através da *aplicação web* do DAAV e foram armazenadas, com a ajuda do componente *server*, na tabela *apk2scan* do componente *database* e ficando a aguardar a atuação do componente *manager*.

Ao ser executado através do temporizador (processo "*cron*" do Linux), o componente *manager* demorou cerca de 40 minutos para receber dos *plugins* com resultados de análise dos 20 APKs, gerando 80 relatórios para visualização do utilizador, destes relatórios gerados, 60 são a simples reposta individual de cada *plugin* sobre cada APK, e os outros 20 são o resultado do trabalho de normalização e categorização dos outros 60 que o *manager* executa e retorna incluindo *feedbacks*. Portanto para cada APK são

disponibilizados três relatórios individuais dos *plugins* e um categorizado com a soma dos outros três.

Os componentes do DAAV, para o estudo de caso em questão, foram executados em uma máquina com o sistema operativo Windows 10, com um processador de 1.80Ghz da Intel (i7-8550U), 16 Gb de memória RAM e um disco rígido de 250 Gb.

O resultado da categorização pela metodologia OWASP mostrou que, nas aplicações analisadas, praticamente as mesmas vulnerabilidades foram encontradas, e a soma de todas as vulnerabilidades encontradas nas 20 aplicações são apresentadas na *Tabela 9.*

M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
318	218	166	0	40	0	59	63	40	77

Tabela 9. Número de vulnerabilidades encontradas nos 20 APKs deste estudo.

Nota-se que de todas aplicações analisadas, não foram encontradas por nenhum *plugin* vulnerabilidades do tipo M4 ou M6, ou seja, autenticação e autorização de utilizadores são preocupações de todos os programadores envolvidos nestas aplicações. Os outros valores são pouco expressivos se levar em consideração que muitas vulneravilidades se repetem por estas aplicações, a *Tabela 10* apresenta os números de vulnerabilidades categorizadas para cada aplicação analisada neste estudo de caso, seguindo a ordenação proposta na *Tabela 8.*

Aplicação	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
Clash Royale	16	10	9	0	2	0	3	3	2	4
Facebook	16	11	8	0	2	0	3	3	2	4
Instagram	16	11	8	0	2	0	3	3	2	4
Messenger	16	11	8	0	2	0	3	3	2	4

Polarr Photo Editor	15	10	8	0	2	0	3	4	2	4
Torch Flashlight LED HD	16	10	9	0	2	0	3	2	2	3
WhatsApp Messenger	16	11	8	0	2	0	3	3	2	4
Clash of Clans	16	12	8	0	2	0	3	4	2	3
Pokemon Go	15	10	8	0	2	0	2	3	2	4
Snapchat	16	11	8	0	2	0	3	3	2	4
Wish	16	12	9	0	2	0	3	3	2	4
Dropbox	16	12	9	0	2	0	3	3	2	4
Hangouts	16	11	8	0	2	0	3	3	2	4
Netflix	16	11	8	0	2	0	3	4	2	4
Sixt Car Rent	16	10	10	0	2	0	3	3	2	4
Spotify	16	11	8	0	2	0	3	3	2	4
VLC for Android	15	11	8	0	2	0	3	3	2	4
Waze	16	11	8	0	2	0	3	3	2	4
Uber	17	11	8	0	2	0	3	4	2	4
LinkedIn	16	11	8	0	2	0	3	3	2	3

Tabela 10. Número de vulnerabilidades encontradas em cada APK deste estudo.

As aplicações analisadas compartilham praticamente as mesmas vulnerabilidades encontradas pelo sistema DAAV, mas dos números vistos na *Tabela 10*, é ainda possível categorizar as vulnerabilidades em nível de severidade, a *Tabela 11* percorre as mesmas aplicações, mas apresenta a quantidade de vulnerabilidades separadas pelos níveis, do menor para o maior, nomeadamente *Informação, Atenção* e *Crítico.* Estas informações

de nível de severidade estão disponíveis no relatório detalhado que o DAAV apresenta de cada APK analisado.

Aplicação	Informação	Atenção	Crítico
Clash Royale	40	6	3
Facebook	41	5	3
Instagram	41	5	3
Messenger	41	5	3
Polarr Photo Editor	40	4	4
Torch Flashlight LED HD	40	3	4
WhatsApp Messenger	40	6	3
Clash of Clans	41	7	3
Pokemon Go	41	2	3
Snapchat	41	6	2
Wish	40	7	4
Dropbox	42	6	3
Hangouts	40	6	3
Netflix	41	6	3
Sixt Car Rent	40	6	4
Spotify	40	6	3
VLC for Android	40	5	3

Waze	41	5	3
Uber	41	5	5
LinkedIn	40	5	3

Tabela 11. Número de vulnerabilidades separadas por nível de severidade.

Com os dados apresentados nas tabelas *9, 10* e *11* a ferramenta DAAV automaticamente gera as estatísticas na página principal para visualização do utilizador, conforme *Figura 21*.

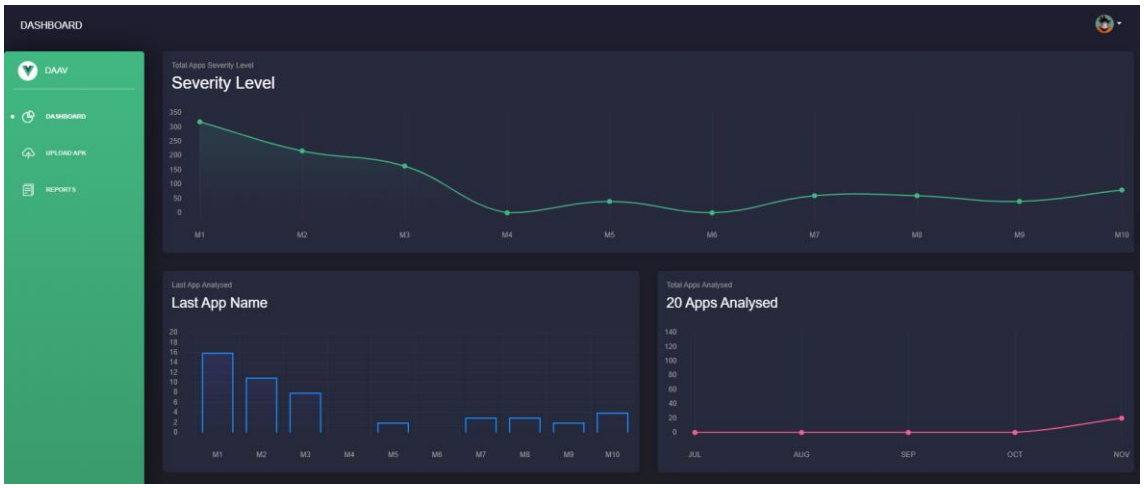

Figura 21. Gráficos gerados pelo DAAV neste estudo de caso.

Para efeitos deste estudo de caso, será escolhida uma aplicação randômicamente deste grupo de 20 APKs para análise dos resultados e busca de solução, conforme proposta da ferramenta, e para facilitar a compreensão desta análise, foi escolhida as vulnerabilidades de categoria M3, pois basicamente uma falha desta categoria é corrigida com a utilização de um protocolo de segurança nas comunicações com o servidor.

Se o programador que criou esta aplicação tivesse utilizado a ferramenta DAAV, encontraria o trecho apresentado na *Figura* 22 do resultado do relatório de vulnerabilidades categorizado de sua aplicação.

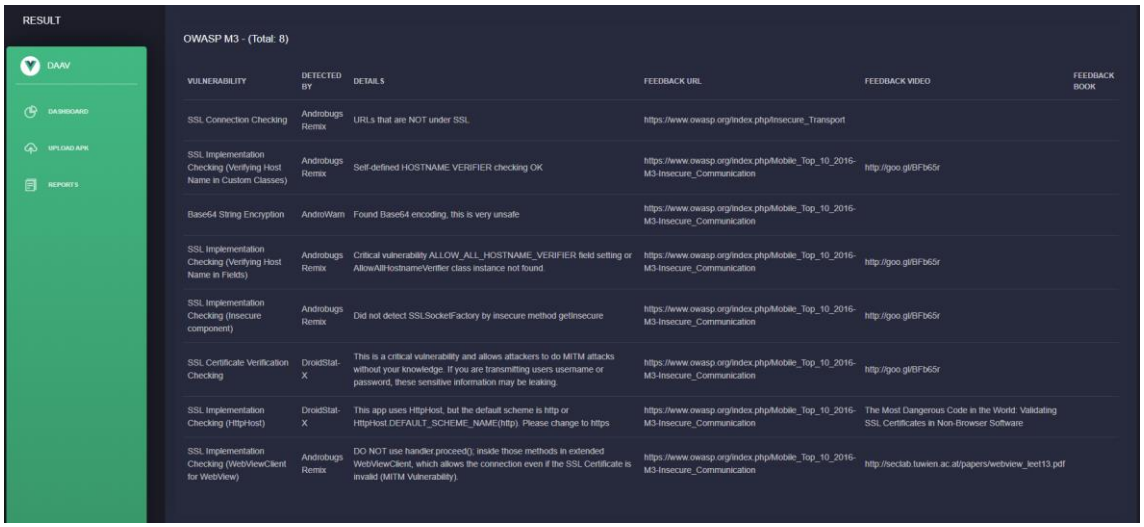

Figura 22. Trecho do relatório categorizado de uma aplicação deste estudo de caso.

Uma das vulnerabilidades descritas na *Figura 22* é a encriptação de alguma *string* em Base64, que é considerada muito insegura, ao lado desta informação um *link* para o *website* da OWASP que indica o que isso significa e ainda a sugestão de como substituir essa encriptação por outra mais segura. Ao aplicar a sugestão do DAAV, esta menção no relatório desapareceria e a aplicação tornaria-se mais segura.

O *feedback* que a ferramenta DAAV apresenta, são *links* para *websites* externos que se relacionam a vulnerabilidade encontrada, e contém informações para solucioná-la. Comumente apontam para *websites* como a OWASP que, como descrito no ítem **2.2** do segundo capítulo, contém um grande acervo de documentos sobre segurança para programadores, mas não se limita a isso. O DAAV tem também armazenado no componente *database*, *links* para vídeos ou livros que possam ajudar ainda mais o programador a perceber a vulnerabilidade encontrada, deixando-os mais conscientes para a preocupação em segurança durante a criação da aplicação.

Se levar em consideração que vinte aplicações foram avaliadas pelo sistema DAAV, e que em menos de uma hora, o utilizador recebe relatórios devidamente classificados com indicações de solução para cada vulnerabilidade encontrada, é possível afirmar que a utilização do DAAV durante a criação de uma aplicação, pode de forma relativamente rápida, ajudar a reduzir as falhas de segurança, mesmo que o utilizador não tenha grande conhecimento nesta área, afinal o relatório indica no *feedback* todo o conhecimento necessário para resolver a falha encontrada com detalhes.

Conclui-se com este estudo de caso, que a fácil e rápida utilização do DAAV, somado a confiabilidade da ferramenta comparada as outras existentes, pois as contém dentro de si, e a categorização em um formato padrão que facilita a visualização dos problemas encontrados no código, e ainda, a indicação de resposta as falhas localizadas, fazem com que o programador que utilize o sistema DAAV, crie uma aplicação mais segura, mesmo sem grandes conhecimentos de segurança e boas práticas de código. A criação de aplicações mais seguras, implicam na redução de *malwares*, que não conseguirão encontrar falhas para se instalar, e consequentemente trazem maior segurança aos dados do utilizador final em seu dispositivo móvel.

Capítulo 5 – Conclusões e Trabalho Futuro

Este capítulo tem o intuito de apresentar as conclusões após estudo, criação e resultados da ferramenta DAAV, o que pode se esperar para o futuro desta ferramenta, e também apresentar sugestões para trabalhos futuros a partir deste.

5.1. Conclusões

O sistema DAAV obteve sucesso na questão de reunir diversas ferramentas de análise de código em sistemas Android e apresentar ao utilizador um relatório categorizado e com *feedback* das falhas de segurança, e, por unir diversas ferramentas em apenas uma, já comprova sua vantagem sobre as que estão a atuar em separado.

Este trabalho também demonstrou que a melhor forma de manter os dados do utilizador de dispositivos móveis seguro, é a utilização de aplicações que não contenham falhas de segurança a serem exploradas por *malwares*, portanto, o papel do programador é de extrema importância na criação deste tipo de aplicações.

A falta de interesse dos programadores com segurança de software, proveniente do curto tempo para desenvolver uma aplicação, ou falta de conhecimento, ou dificuldade em encontrar tais falhas, é o principal problema de segurança que o utilizador enfrenta ao instalar aplicações em seu dispositivo, e o sistema DAAV tem o objetivo de ajudar neste sentido. Com a facilidade de acesso em comparação as complexas configurações das ferramentas existentes de análises, a credibilidade dos resultados pelo fato de unir diversas ferramentas, e ainda, pela indicação de resolução das vulnerabilidades, espera-se que os programadores construam softwares de melhor qualidade e livre de falhas de segurança ao aplicar o sistema DAAV em sua criação de código.

Esta nova ferramenta ainda tem muito a crescer, como por exemplo com o envio massivo de APKs para teste, criação de um relatório mais agradável para visualização ou inclusão de mais ferramentas de análise como *plugin*. E como a ferramenta foi baseada em aplicações de código-aberto, também se encontra disponível em um repositório *online* (https://github.com/rasc-br/DAAV).

Este trabalho foi desenvolvido de forma integrada no projeto AppSentinel, co-financiado pelo Lisboa2020/Portugal2020/EU no contexto do Sistema de Incentivos à I&DT - Projetos em Copromoção (project 33953).

5.2. Trabalhos Futuros

Para que os utilizadores tenham seus dados seguros em dispositivos móveis, este trabalho foca na idéia de que as aplicações instaladas em tais dispositivos devem ser seguras, ao invés de criar métodos para impedir os ataques. Portanto os programadores devem se preocupar mais com a segurança e criar aplicações seguras, evitando assim qualquer tipo de ataque.

Mesmo que focado em proteger a aplicação, o programador não conseguiria sozinho bloquear todas as possíveis falhas de segurança, portanto, ferramentas que ajudem na busca dessas falhas são essenciais. Neste sentido a ferramenta DAAV é uma vantagem na luta contra a falta de segurança em dispositivos móveis.

Este trabalho deixa como sugestão para trabalhos futuros:

- A pesquisa para encontrar o motivo da falta de interesse dos programadores em criar métodos de segurança sem seu código.
- Descobrir a dificuldade dos programadores em encontrar e solucionar as falhas de segurança em seu código.
- Um apriporamento da ferramenta de análise aqui proposta, no intuito de facilitar ainda mais a solucionar as falhas de segurança durante a criação do código da aplicação.

Bibliografia

André, C. (2019). GitHub clviper/droidstatx. Retrieved October 28, 2019, from Github website: https://github.com/clviper/droidstatx

Aptoide Team. (2019). Aptoide | Download, Find, Share the Best Android Games and Apps. Retrieved October 29, 2019, from http://aptoide.com/en/home

Blyth, A. (2004). Innocent code: a security wake-up call for web programmers. *Infosecurity Today*, *1*(4), 43. https://doi.org/10.1016/s1742-6847(04)00092-8

Bose, A., Hu, X., Shin, K. G., & Park, T. (2008). Behavioral detection of malware on mobile handsets. *MobiSys'08 - Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, (September), 225–238. https://doi.org/10.1145/1378600.1378626

Botha, R. A., Furnell, S. M., & Clarke, N. L. (2009). From desktop to mobile: Examining the security experience. *Computers and Security*, *28*(3–4), 130–137. https://doi.org/10.1016/j.cose.2008.11.001

Chandramohan, M., & Tan, H. B. K. (2012). Detection of mobile malware in the wild. *Computer*, *45*(9), 65–71. https://doi.org/10.1109/MC.2012.36

Cohen, F. B., Highland, H. J., & David, J. (1991). A short course on computer viruses. *Computer Fraud & Security Bulletin*, *1991*(1), 17–18. https://doi.org/10.1016/0142-0496(91)90211-m

Collberg, C. S., & Thomborson, C. (2002). Watermarking, tamper-proofing, and obfuscation - Tools for software protection. *IEEE Transactions on Software Engineering*, *28*(8), 735–746. https://doi.org/10.1109/TSE.2002.1027797

Cova, M., Kruegel, C., & Vigna, G. (2010). Detection and analysis of drive-by-download attacks and malicious JavaScript code. *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, 281–290. https://doi.org/10.1145/1772690.1772720

Cox, V. (2018). Github - vincentcox/StaCoAn. Retrieved October 28, 2019, from Github website: https://github.com/vincentcox/StaCoAn

Cromwell, V. (2016). Between the Wires interview | Evan You. | Between the Wires. Retrieved October 29, 2019, from Between the Wires website: https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/

Dadax. (2019). World Population Clock: 7.7 Billion People (2019) - Worldometers. Retrieved October 28, 2019, from Current world population website: https://www.worldometers.info/world-population/

Debize, T. (2013). maaaaz/androwarn: Yet another static code analyzer for malicious Android applications. Retrieved October 28, 2019, from Github website: https://github.com/maaaaz/androwarn

Desnos, A., & Gueguen, G. (2011). Android: From reversing to decompilation. *Proc. of Black Hat Abu Dhabi*, 1–24.

Douglas, C. (2019). Introducing JSON. Retrieved October 29, 2019, from http://www.json.org/

Dunham, K., Hartman, S., Morales, J. A., Quintans, M., & Strazzere, T. (2014).

Android malware and analysis. In *Android Malware and Analysis*. https://doi.org/10.1201/b17598

Foundation, P. S. (2016). *The Python Language Reference*. 8–9. Retrieved from https://docs.python.org/3/reference/

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly: Management Information Systems*, *28*(1), 75–105. https://doi.org/10.2307/25148625

Howard, M. (2004, November). Building more secure software with improved development processes. *IEEE Security and Privacy*, Vol. 2, pp. 63–65. https://doi.org/10.1109/MSP.2004.95

Hypponen, M. (2006). *Malware Goes Mobile*.

ibotpeaches. (2016). Apktool - Documentation. Retrieved October 28, 2019, from Octubre website: https://ibotpeaches.github.io/Apktool/documentation/

Kramer, S., & Bradfield, J. C. (2010). A general definition of malware. *Journal in Computer Virology*, *6*(2), 105–114. https://doi.org/10.1007/s11416-009-0137-1

Kuechler, B., & Petter, S. (2012). *D Esign S Cience R Esearch in*. (1), 1–66. https://doi.org/1756-0500-5-79 [pii]\r10.1186/1756-0500-5-79

Lacerda, D. P., Dresch, A., Antunes Júnior, J. A. V., & Proença, A. (2013). Design Science Research: A research method to production engineering. *Gestao e Producao*, *20*(4), 741–761. https://doi.org/10.1590/S0104-530X2013005000014

Lin, J., Amini, S., Hong, J. I., Sadeh, N., Lindqvist, J., & Zhang, J. (2012). *Expectation and Purpose: Understanding Users' Mental Models of Mobile App Privacy through Crowdsourcing*.

Lin, Y. (2015). *Androbugs Framework: An Android Application Security Vulnerability Scanner*. Retrieved from http://www.androbugs.com

LinkedIn. (2019a). linkedin/qark: Tool to look for several security related Android application vulnerabilities. Retrieved October 28, 2019, from Github website: https://github.com/linkedin/qark

LinkedIn. (2019b). LinkedIn Github Profile. Retrieved October 28, 2019, from Github website: https://github.com/linkedin

Lopes, J., Serrão, C., Nunes, L., Almeida, A., & Oliveira, J. (2019). Overview of machine learning methods for Android malware identification. *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, 1–6.

McGraw, G. (2002). Building Secure Software: Better than Protecting Bad Software. *IEEE Software*, *19*(6), 57–58. https://doi.org/10.1109/MS.2002.1049391

McGraw, G. (2008). Automated code review tools for security. *Computer*, *41*(12), 108–111. https://doi.org/10.1109/MC.2008.514

Möller, A., Michahelles, F., Diewald, S., Roalter, L., & Kranz, M. (2012). Update Behavior in App Markets and Security Implications : A Case Study in Google Play. *Proceedings of the 3rd International Workshop on Research in the Large. Held in Conjunction with Mobile HCI*, 3–6. Retrieved from http://www.backes-srt.de/produkte/srt-appguard

Nachenberg, C. (1997). Computer virus-antivirus coevolution. *Communications of the*

ACM, *40*(1), 46–51. https://doi.org/10.1145/242857.242869

Narudin, F. A., Feizollah, A., Anuar, N. B., & Gani, A. (2016). Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, *20*(1), 343–357. https://doi.org/10.1007/s00500-014-1511-6

OWASP. (2014). OWASP Mobile Top Ten. Retrieved October 28, 2019, from https://www.owasp.org/index.php/OWASP_Mobile_Top_10

OWASP Foundation. (2015). Category:OWASP Top Ten Project - OWASP. *OWASP™ Foundation*. Retrieved from https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Pachev, A., & Pachev, S. (2007). *Understanding MySQL internals* (Vol. 0). Retrieved from http://books.google.com/books?id=vz6PcTdo8VUC&pgis=1

Robert, J.-M., & Chen, T. (2004). *The Evolution of Viruses and Worms*. 265–285. https://doi.org/10.1201/9781420030884.ch16

Rossum, G. Van, & Drake, F. L. (2012). Porting Python 2 Code to Python 3. *Strategy*, 3–8.

Sandbox, C. (2011). *Cuckoo Sandbox Book*. Retrieved from https://cuckoo.sh/docs/

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G., & Álvarez, G. (2013). PUMA: Permission usage to detect malware in android. *Advances in Intelligent Systems and Computing*, *189 AISC*, 289–298. https://doi.org/10.1007/978-3-642-33018-6_30

Seghir, M. N., & Aspinall, D. (2015). EviCheck: Digital evidence for android. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9364*, 221–227. https://doi.org/10.1007/978-3-319-24953-7_17

SOTIROV, A. I. (2005). Automatic vulnerability detection using static source code analysis. *Vasa*, 1–118. Retrieved from http://medcontent.metapress.com/index/A65RM03P4874243N.pdf%5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.9646&rep=rep1&type=pdf

Statista. (2016). *Mobile app usage - Statista Dossier*. Retrieved from https://www.statista.com/topics/1002/mobile-app-usage/

Statista. (2018). Annual number of mobile app downloads worldwide 2022 | Statistic. Retrieved October 28, 2019, from Statista Web Site website: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/

Statista. (2019). Mobile OS market share 2018. Retrieved October 29, 2019, from statista website: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

Tenenboim-Chekina, L., Barad, O., Shabtai, A., Mimran, D., Rokach, L., Shapira, B., & Elovici, Y. (2013). *Detecting Application Update Attack on Mobile Devices through Network Features*. https://doi.org/10.1109/INFCOMW.2013.6970755

U.S. Department of Homeland Security. (2019). Homepage | CISA. Retrieved October 28, 2019, from https://www.us-cert.gov/

Wolfgang Keller, J. C. (2019). A Relational Database Overview (The Java™ Tutorials

> JDBC(TM) Database Access > JDBC Introduction). Retrieved October 29, 2019, from https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

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