UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

AUTOMATIZAÇÃO DE REQUISITOS DE SEGURANÇA EM APLICAÇÕES ANDROID

RUI FILIPE GAGO PEREIRA
MESTRADO EM SEGURANÇA INFORMÁTICA

Dissertação orientada por:
Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves
Prof. Doutora Ibéria Vitória de Sousa Medeiros

2020

Agradecimentos

Esta dissertação é fruto de um projecto académico, que não teria sido possível realizar sem o apoio directo e indirecto de algumas pessoas, as quais passo a mencionar. Em primeiro lugar, quero agradecer ao Professor Doutor Nuno Neves e à Professora Doutora Ibéria Medeiros, por terem aceite este desafio, pelo seu profissionalismo, dedicação e disponibilidade, e pela motivação que me deram no decorrer desta dissertação. Quero também agradecer à minha namorada, pela sua paciência, compreensão e apoio que me deu nesta fase da minha carreira, e que se mostrou fundamental para que não perdesse o foco dos meus objectivos. Agradeço também à minha mãe e à minha avó Lucília, por apoiarem-me na minha vida académica e zelarem pelo meu sucesso. Um agradecimento especial para todos os meus colegas e amigos que me acompanharam e motivaram nesta jornada, destacando os seguintes nomes: André Corda, João Palma, Manel Oliveira, Bruno Batista, Gustavo Correia, Gonçalo Correia, Rodrigo Dias, Luís Garcia, João Marques e Miguel Chaves. Um grande obrigado a todos.

Ao meu filho

Resumo

Na sociedade actual é difícil de identificar alguém que não use um *Smartphone*. São várias as vantagens que estes pequenos dispositivos móveis trazem, incluindo um mercado de aplicações que nos facilita a vida. Hoje em dia, o nosso telemóvel é um espelho do computador de casa, contendo por vezes uma mistura de informação pessoal e informação profissional.

Estes dispositivos têm uma ligação para um repositório que oferece várias aplicações que se podem instalar, mas para que estas aplicações sejam disponibilizadas no mercado não é obrigatório que o seu código implemente e cumpra requisitos de segurança. Esta falta de obrigatoriedade tem levado a que todos os anos sejam descobertas vulnerabilidades em aplicações, sejam *Android*, *IOS* ou *Windows*, que colocam em causa os dados pessoais dos utilizadores. Por exemplo, em 2019 a Forbes anunciou que a Uber assumiu a existência de uma vulnerabilidade na sua aplicação que permitia ao atacante ter controlo total da conta, incluindo verificar quais os trajectos que este já havia feito ou até mesmo os seus dados bancários associados.

Para qualquer solução informática existe um conjunto de requisitos de segurança que devem ser verificados, e as aplicações *Android* não são excepção. Esta dissertação foca-se na criação do ransAPK, um sistema com a capacidade de verificar automaticamente requisitos de segurança de uma aplicação *Android* e medir a sua maturidade com base nos requisitos que esta assegura. O ransAPK usa uma metodologia apresentada pela comunidade OWASP, mas executa-a de forma autónoma.

Para prova de conceito, o ransAPK foi executado sobre 17 aplicações de três tipos: bancárias, rastreio do COVID-19 e com código protegido (*Packed*). Os resultados obtidos indicam que nenhuma das aplicações cumpre os requisitos na sua totalidade, o que coloca os utilizadores das mesmas em risco, se as vulnerabilidades que elas contêm forem exploradas.

Palavras-chave: Vulnerabilidades, Aplicações *Android*, Qualidade de código, Automação de verificação de requisitos, Análise de requisitos de segurança

Abstract

The number of global smartphone users, currently, has reached approximately 4 billion. Smartphone users rely on their devices to store private and personal data, including photos, contacts, mail, and bank account information. However, security and data breaches have accrued and increased in smartphones, witch has led to severe financial, social, and psychological consequences. Mobile security mechanisms have been in continuous improvement in the last 10 years, with the addition of multiple software and hardware security layers. For example, nowadays the usage of passwords, screen locks and biometric information, such as fingerprint data, is being increasingly recommended. But smartphone operating systems and software frameworks are relatively recent and in continuous development. Hence, the security of smartphones is still at an early stage. Meanwhile, the level of sophistication and craftiness of malware is constantly evolving, which is encouraged by the high rewards, the immaturity of security countermeasures, and the sheer number of users and potential victims.

The primary source of system vulnerabilities and entry points for malware attacks in a modern smartphone is the installation of insecure 3rd party applications by an inconspicuous user. The associated issues are mainly caused by bad, or even the absence of security practices used during application development. For example, a 2018 corporate survey of 17 Android apps discovered a high prevalence of critical vulnerabilities, including vulnerabilities on the client side which could be exploited without administrator rights. Therefore, it is of paramount importance to improve the security practices used in mobile app development and incorporate security requirements into the Software Development Life Cycle (SDLC). In an effort to address this need, the OWASP community created a methodology to validate the fulfilment of security requirements implemented in an Android application. Briefly, there are eight main sets of requirements, grouped according to their domain/theme, which should be met by an application to be certified as ”secure”. These are Architecture Design and Threat Modeling, Data Storage and Privacy, Cryptography, Authentication and Session Management, Network Communication, Platform Interaction, Code Quality and Build Setting and Resilience. The Architecture Design and Threat Modeling requirements are concerned with the architecture and design of the application, and cover topics such as SDLC, user access control, session key management, and threat modelling. Data Storage and Privacy requirements focus on validating the

safety practices of the information contained in an application and which is stored in the device. The Cryptography group specifies and validates the need for adequate choice and implementation of cryptographic algorithms. Authentication and Session Management focuses on validating whether the session tokens and authentication processes are adequately protected, and transactions are performed safely. The Network Communications checks the security of the communication between the application and external servers. Requirements of the Platform Interaction group have the aim of validating the safety of typical user and application interactions. Code Quality and Build Setting requirements promote the adherence to good Android programming practices. Finally, the Resilience group aims to ensure that the intellectual property of the organisation that created the application is adequately protected. In total, there are 72 individual requirements, each one belonging to one of the eight groups. Ideally, an application should meet all 72 to guarantee an acceptable level of safety for the user. One of the main barriers for the adoption of the OWASP guidelines in everyday software development is the high number of requirements and checklists, whose implementation depends on modification of coding practices and requires manual validation from trained security analysts. Most security tools and analysis software are tailored for the detection of system and application vulnerabilities in existing applications and are therefore corrective instead of preventive. In contrast, the offer of open source software tools which focuses on the promotion of secure practices and help development teams implement correctly (including automated assessment of their fulfilment) the OWASP or other guidelines is very scarce.

To address this unmet need, this dissertation presents the development of a novel software system, ransAPK, capable of automatically validating whether the security requirements laid out in the OWASP community guidelines are being met during the application development process. RansAPK is implemented in Python and is accessed via an interactive web graphical interface. It requires connection to either a physical or virtual mobile device via the Android Debug Bridge (adb). The ransAPK system uses static and dynamic analysis to inspect the app's source code and validate automatically the fulfilment of 34 out of the 72 OWASP requirements. To guide the implementation of the remaining 38 requirements, whose automation is not straightforward due to necessity of user input or other complex interactions, ransAPK provides the user with an interactive version of the requirements checklist via its web interface. RansAPK also calculates the Mobile Application Security Verification (MASV) score based on the requirement fulfilment status.

RansAPK's source code analysis starts by inspecting whether the app contains protected source code (packed app) or not. In the latter case, the .APK file is decompiled using the apktool to obtain the app's .dex file with its Java bytecode. The instructions set contained in this file are then translated to a higher-level language, Java, using dex2jar. The resulting files are then submitted to a static code analysis process, which is performed using the Yara tool. The tool is fed with Yara rules, which encode the OWASP

requirements that allow their automated verification. If the app's source code is protected, ransAPK employs dynamic code analysis instead. For that, the app is automatically installed and executed on the mobile device, and its bytecode retrieved from memory at runtime, using the Frida dynamic instrumentation toolkit. The derived .dex file is then translated to Java and analysed with the Yara tool.

The user interaction with ransAPK is handled primarily through the web interface. Here, the analyst can directly upload an .APK file for analysis through drag and drop. Multiple applications can be analysed at the same time, and ransAPK will keep a list of the results in an sqllite database, even after terminating the session. For each app analysed, a summary radar chart is displayed, where each vertex corresponds to the score of a particular OWASPs requirement group. Both results of automated and manual requirement validations are aggregated and used to calculate the MASV score, which is also shown next to the radar chart. This score is automatically recalculated if the number of completed requirements is changed.

To demonstrate and test the ransAPK functionality, seventeen mobile banking, COVID-19 track and trace, and one "Packed" applications were analysed. None of the twelve mobile banking app completely met the OWASP guidelines. In seven of them, the ransAPK analysis highlighted severe flaws in the implementation of requirements from the Resilience group. These are particularly problematic for banking apps, since vulnerabilities in this sensitive area can enable attackers to modify the app's code at runtime and inject malicious instructions, which can be used to divert a bank transfer to an unwanted recipient. ransAPK also highlighted a poor implementation of the OWASP Network Communication requirements in some of the banking apps. Some of the vulnerabilities discovered could allow the interception of communications between server and client, enabling man-in-the-middle attacks.

For COVID-19 applications, the Resiliency and Network and Communication groups were also the main areas which require extensive improvement. Three out of the four applications surveyed did not implement correctly four out of a total of seven Resilience requirements, which increases the likelihood of the application being vulnerable to injection of malicious data at runtime. Interestingly, two of the COVID-19 apps tested with ransAPK had poor compliance with Network and Communication requirements, which implies they can be vulnerable to man-in-the-middle attacks. This type of attack can be used to de-anonymise the user identity. Given the effort in the development of completely anonymised track and trace, and the potential loss of public trust in COVID-19 apps that fail to protect user's privacy, these findings warrant further investigation and improvement of the security profile of these apps. With the analysis of the Packed application we showed that ransAPK was able to analyse application with their source code protected.

In summary, ransAPK showed the potential to facilitate the automation and verification of OWASP's guidelines compliance, by lowering the effort by the analyst up to 50%.

Besides these gains from automation, ransAPK also promotes the full implementation of the OWASP security methodology in the software development pipeline by providing users with interactive summaries and checklist with the overview of an application's compliance and the corresponding MASV score. An advantage of ransAPK, when compared with other security tools, is that both static and dynamic code analysis workflows are implemented in order to automate the analysis of a larger number of applications at same time. Further improvements to ransAPK could include the reinforcement of the current Yara rule set to enhance the coverage of the automated requirement validations, and improvements to the graphical user interface, such as the addition of links to documentation and example cases to provide technical guidance to the user on how to fulfil a particular requirement.

Keywords: Vulnerabilities, Android applications, Code quality, Automatic requirements validation, Security requirements analysis

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Actualmente os *smartphones* tornaram-se parte das nossas vidas e revolucionaram a maneira como realizamos a maioria das nossas actividades. O que era realizado fazer num computador ligado à Internet, hoje podemos fazê-lo de qualquer lugar, bastando apenas um acesso à Internet e a aplicação correcta instalada no nosso dispositivo. Por exemplo, se quisermos verificar o saldo bancário, instalamos a aplicação do nosso banco e com as credências correspondentes temos acesso à conta. O mesmo acontece se quisermos verificar os e-mails da nossa organização. Instalamos a aplicação correspondente e temos acesso a todos os documentos contidos na nossa caixa de correio, permitindo-nos até que sejam guardados estes documentos no nosso dispositivo. Em resumo, um *smartphone* agora é um enorme repositório que contém informações confidenciais e pessoais sobre o seu proprietário. Para qualquer operação que queiramos realizar, existe uma aplicação, e quantas mais operações realizarmos com estes dispositivos mais informação confidencial estará contida no dispositivo.

Actualmente o mercado da Google disponibiliza na sua loja perto de 3 milhões de aplicações, onde aproximadamente 96% são gratuitas e 4% requerem um pagamento para que haja acesso à aplicação [23]. Esta popularidade dos *smartphones Android* atraíram uma grande variedade de ciber ataques. Alguns envolvem técnicas avançadas de *debug*, em que o atacante altera em tempo real os valores de memória onde as variáveis de execução estão armazenadas, para seu benefício [7]. Alguns exploram defeitos endógenos nos programas *Android*, ou seja, se o atacante descobre que uma determinada biblioteca usada num APK (aplicação *Android*) está vulnerável, pode usar esse recurso para identificar outras aplicações que estão vulneráveis por fazerem uso da mesma. Existem também aplicações que encobrem o roubo de identidade com camuflagem: as aplicações apresentam semelhanças à aplicação normal, mas na realidade são uma cópia com intuito malicioso, permitindo ao atacante a possibilidade de executar vários comandos sem que o utilizador tenha conhecimento. Um grupo de investigadores *Android*, MalwareHunterTeam, detectou que o mercado da Google continha várias aplicações que usavam esta técnica para obter dados pessoais dos utilizadores [18] .

Para derrotar tentativas maliciosas, os analistas de segurança de aplicações *Android* contra-atacam. Muitas técnicas tradicionais foram estudadas, existindo uma classificação atribuída pela Google por 13 classes distintas [14]: *Backdoor*, *Billing fraud*, *Commercial spyware*, *Denial of service*, *Hostile downloaders*, *Phishing*, *Non-Android threat*, *Elevated privilege abuse*, *Ransomware*, *Rooting*, *Spam*, *Spyware* e *Trojan*. Uma aplicação que realize pelos menos uma destas técnicas é chamada, genericamente, de Malware. Outras duas práticas também estudadas são a análise de propagação de Malware, onde é possível traçar um perfil através das acções maliciosas e agrupar tipos de Malware por famílias. Outra prática é em relação ao controlo de acesso do dispositivo, que nos últimos anos tem evoluído de versão para versão. O controlo de acesso actualmente é feito por reconhecimento facial ou por impressão digital. No entanto, as técnicas intrusivas têm evoluído e, infelizmente, as defesas disponíveis ficam aquém, fundamentalmente devido à complexidade das versões de *Android* existentes.

Para resolver esta limitação foi necessário observar o problema de uma outra perspectiva. Aplicações *Android*, não importa se boas, más ou vulneráveis, são de facto programas de *Software* e estes programas devem assegurar requisitos de segurança.

Se estudarmos o ciclo de vida de uma aplicação até ser divulgada no mercado Google Play (o mercado oficial de aplicações *Android*), não está contémplada a existência de testes de intrusão. A Google Play Protect foca-se exclusivamente em detectar aplicações com *Malware* ou de conteúdo impróprio para o seu público. As vulnerabilidades de uma aplicação são da responsabilidade de quem desenvolve a mesma. Logo, os requisitos de segurança de uma aplicação acabam por ser definidos por quem a produz, sendo este o responsável pelas suas validações.

São várias as vulnerabilidades descobertas ao longo da história das aplicações para *Android*. Por exemplo, recentemente foi encontrada uma vulnerabilidade no *Instagram* que permitia ao atacante ter acesso total ao telemóvel da vítima [25]. Também, cerca de 85% das aplicações responsáveis por controlar a propagação do SARS-CoV-2 estavam vulneráveis a um ataque que permitia identificar e rastrear o utilizador. Os utilizadores que usam estas aplicações confiam-lhes cada vez mais os seus dados pessoais, mas quem as desenvolve não tem qualquer obrigação de defender o consumidor. Para se perceber o risco em que os utilizadores estão expostos é necessário conhecer as vulnerabilidades existentes em aplicações móveis.

Segundo a comunidade OWASP - *"Open Web Application Security Project"* [29], o Top 10 das vulnerabilidades *mobile* são:

- M1: Uso de Plataforma impróprio
- M2: Armazenamento de dados inseguro
- M3: Comunicação Insegura
- M4: Autenticação Insegura

- M5: Criptografia Insuficiente
- M6: Autorização Insegura
- M7: Qualidade de código
- M8: Adulteração de código
- M9: Engenharia inversa
- M10: Funcionalidade estranha

Seguindo esta lista, estes são os tópicos principais que se deve ter em conta quando é necessário impor requisitos de segurança a uma aplicação *Android*. Esta dissertação apresenta uma solução simples e prática para a validação deste tipo de requisitos, seguindo as recomendações da comunidade OWASP, para concretizar estes objectivos, verificando assim quanto uma aplicação é segura. Foi construída um sistema com a capacidade de validar quais os requisitos são implementados por uma aplicação e quais estão em falta. O sistema possibilita que qualquer APK seja analisado em alguns minutos e apresenta ao utilizador uma visão de quais as áreas precisam de ser melhoradas para garantir uma menor cobertura de vectores de ataque.

O nome deste sistema é ransAPK e, após ter sido executado em várias aplicações bancárias e aplicações de rastreio de COVID-19, foi possível identificar que algumas destas aplicações não estão em total conformidade, o que pode indicar a existência de vulnerabilidades que põem em risco os seus utilizadores.

1.1 Motivação

As aplicações *mobile* têm um universo pouco explorado de soluções que auxiliam a sua construção de modo seguro. Para além do pouco material que actualmente existe para investigar um APK, são poucas as ferramentas que se mantêm actualizadas. Para além disso, os programadores na sua maioria não conhecem boas práticas no processo de desenvolvimento de código seguro.

Se analisarmos o estudo da *OWASP Mobile Security Project* [29], é visível quais os requisitos necessários para garantir a segurança de uma aplicação. Inclusive é apresentado pela mesma comunidade uma metodologia não optimizada e automatizada, a qual é efectuada manualmente e trabalhada sobre uma folha de cálculo Excel.

São 72 validações que um analista precisa de realizar para garantir a conformidade de segurança de uma aplicação e existe a necessidade de ter um sistema operativo com um leitor de Excel. Se os requisitos forem verificados, é sem dúvida uma mais-valia, e se este processo for feito numa plataforma uniforme, onde mais que um analista pode analisar uma aplicação, e optimizado com mecanismos de validação automáticos para

determinados requisitos, seria tirar várias horas de trabalho a um analista. Estes dois últimos pontos são o foco e a motivação desta dissertação.

1.2 Objectivos

Esta dissertação tem como principal objectivo a criação de um sistema para cálculo de maturidade de uma aplicação. *Android*. Terá como base um processo manual actualmente existente, tornando muitas das tarefas automáticas

Os requisitos a implementar variam devido à criticidade das aplicações, existindo um conjunto finito de requisitos que devem ser observados e usando uma técnica uniforme com o intuito de tornar o trabalho autónomo. Os requisitos de segurança a implementar estão especificados pela comunidade OWASP. Estes serão a base para a construção do sistema. O sistema resultante desta dissertação irá fazer uso das técnicas de análise dinâmica e estática para observar se os requisitos de segurança estão implementados.

De forma a criar uma interacção ágil com o sistema, esta vai disponibilizar uma página *web* que pode ser local ou alojada num servidor web com exposição externa. Este site tem como objectivo ser uma representação gráfica dos resultados e de registo do mesmo, isto porque qualquer requisito pode apenas ter 3 estados *pass*, *not-pass*, *not-applicable*. Ou seja, se um requisito não foi analisado pelo ransAPK pode ser analisado manualmente e registado o seu veredicto nesta página HTTP.

O ransAPK é um sistema de análise de código seguro conforme já descrito e contém a capacidade de analisar aplicações num dos seguintes modos: *black-box*, onde o analista é colocado no papel de atacante, sem nenhuma informação interna da aplicação; *grey-box*, onde o analista tem informação interna sobre o sistema e tem a capacidade de executar funções com utilizadores de privilégios distintos; *white-box*, onde o analista tem todos os perfis disponíveis e tem acesso ao código fonte da aplicação. O sistema não só tem a capacidade de realizar análise estática e dinâmica de uma aplicação, como também é um guia de boas práticas, onde o analista irá ter à disposição um *website* onde pode verificar quais os requisitos que ainda não foram analisados e pode ir marcando manualmente aqueles que o próprio já analisou. O sistema a implementar/construir deve ter as seguintes características:

- Analisar uma aplicação para *Android* seguindo as indicações da comunidade OWASP
- Oferecer ao utilizador uma *Dasboard* para gestão das aplicações, analisadas com o modelo CRUD (*Create, Read, Update e Delete*);
- Oferecer capacidade de instalação simples com o Docker e Docker compose;
- Sistema privado e local.

1.3 Estrutura do documento

Esta dissertação é apresentada com a seguinte organização:

- Capítulo 2 - **análise de conceitos relevantes e estabelecimento de contexto do trabalho a realizar**. O capítulo apresenta um resumo dos conceitos necessários e do trabalho já existente no tema análise de aplicações *Android*. Uma aplicação *Android* não pode ser observada apenas pelo seu comportamento ou pelo código que lhe pertence. É necessário perceber como o Kernel, sistema operativo e as suas máquinas virtuais interagem e possibilitam a execução de uma aplicação conforme vemos no nosso dispositivo.

  Após abordar estes temas será mais fácil a compreensão das vulnerabilidades existentes e como se pode mitigar as ameaças.

- Capítulo 3 - **Arquitectura do *ransAPK*** - Neste capítulo é descrito, em detalhe, a arquitectura do sistema e de cada componente que o comporta.

- Capítulo 4 - **Implementação** - Neste capítulo é explicado em detalhe a implementação do sistema ransAPK, seguindo a arquitectura descrita no capítulo 3.

- Capítulo 5 - **Resultados e Avaliação** - O capítulo apresenta os resultados da avaliação do sistema desenvolvido.

- Capítulo 6 - **Conclusão** - Este capítulo contém um sumário do que foi feito ao longo da dissertação e apresenta algumas conclusões que podem ser retiradas do desenvolvimento do ransAPK.

Capítulo 2

Trabalho relacionado

O sistema desenvolvido neste projecto tem como objectivo analisar uma aplicação *Android* de forma autónoma, suportando a construção de metodologias para validação da maturidade de uma aplicação e atribuir uma classificação. Posto isto, é importante estudar as abordagens ou princípios de abordagem já existentes sobre aplicações de *Android*.

Este capítulo abrange os tópicos fundamentais para a construção do ransAPK. Serão apresentados conceitos técnicos que permitem conhecer o contexto da aplicação *Android* e como esta é executada no sistema operativo. Também, ainda neste tópico, irá ser conhecido o estudo realizado pela comunidade OWASP, estudo este que é a base de construção do ransAPK. Por fim, são descritas ferramentas que apresentam soluções semelhantes ao ransAPK.

2.1 Sistema Operativo *Android*

De forma a compreendermos o nível de ameaça que uma aplicação *Android* está sujeita, é necessário entender o modo de execução e em que arquitectura estas executam. Começando pela arquitectura do sistema operativo *Android*, é possível dividir este em 6 camadas, sendo elas *Aplications*, *Java API framework*, *Libraries*, *Android Runtime*, *Hardware Abstration Layer* e *Linux Kernel* [35]. De entre as diversas razões que levaram à escolha do Linux, a que se destaca é a facilidade com que este sistema operativo se adapta às diferenças de *Hardware*. O *Kernel* actua como uma camada de abstracção entre o *Software* e o *Hardware* presente no dispositivo. Quando existe necessidade de converter instruções de *Hardware* em instruções de *Software*, o *Kernel* contém os *drivers* capazes de facilitar este processo. Por outras palavras, os *drivers* na camada *Kernel*, controlam o *Hardware* correspondente. É também da responsabilidade do *Kernel* a gestão das principais funcionalidades do *Android*, como gestão de processos, a gestão de memória, a segurança e a rede.

Logo acima da camada *Kernel* temos a camada *Hardware Abstration Layer*. Os recursos de *Hardware* de um dispositivo *Android* são expostos através de uma estrutura de

Java de alto nível dentro desta camada. Aqui existem vários módulos de bibliotecas que implementam interfaces para um tipo específico de componente de *Hardware*.

A camada *Libraries*, que contém as bibliotecas em *Android*, é escrita em C e C++ e a sua maioria transita directamente do Linux para *Android*. Uma das maiores diferenças ocorre no uso da biblioteca *Libc*. O *Android* desenvolveu a sua própria biblioteca de nome *bionic* que pode ser vista como uma versão simplificada da *Libc* [8]. Ao mesmo nível da camada de *Libraries* temos a camada de *Android Runtime*, onde podemos encontrar dois tipos de máquinas virtuais para execução de aplicações, a *Android Runtime* e a *Dalvik Virtual Machine*. Após a versão 5.0 da *Android Dalvik Virtual Machine*, esta foi substituída pela *Android Runtime*. Esta alteração deveu-se ao facto de as vantagens do *Android Runtime* serem bastante significativas na velocidade das aplicações, gestão de memória e poupança de energia do próprio dispositivo. O *Dalvik Virtual Machine* executa o *Dalvik Bytecode* que, no fundo, é Java *Bytecode* convertido por um compilador de ficheiros ".Dex".

O *Bytecode* do *Dalvik*, quando comparado com o *Bytecode* gerado pelo Java, é mais adequado para ambientes com pouca memória e baixo processamento. Reparar que contrariamente ao *Java Virtual Machine* (JVM) que distribui o *Bytecode* por vários ficheiros, no *Dalvik* os *Bytecodes* estão todos num ficheiro. A Java *Application Programming Interface* (API) *Framework* é uma camada que oferece um serviço de telefone básico, incluindo gestão de recursos, tratamento de chamadas e outras funções esperadas por este. Esta camada é usada pelas aplicações *Android* para a manipulação dos componentes de um telefone. Por fim, temos a camada *Aplications* que interage com o utilizador, que contém dois tipos de aplicações, as pré-instaladas e as instaladas pelo utilizador [39].

As aplicações pré-instaladas são definidas pelos comerciantes, e incluem um conjunto que serve de aplicações para a interacção básica do utilizador com o dispositivo. As aplicações escolhidas pelos utilizadores podem ser descarregas de várias lojas, mas as principais são a GoogleStore, a Amazon Marketplace e a Aptoide. Para a instalação de aplicações é necessário obter um APK (ver Secção 2.1.1) dos mercados anteriormente referidos. A Figura 2.1 representa graficamente as camadas *Android* e os serviços que estas disponibilizam.

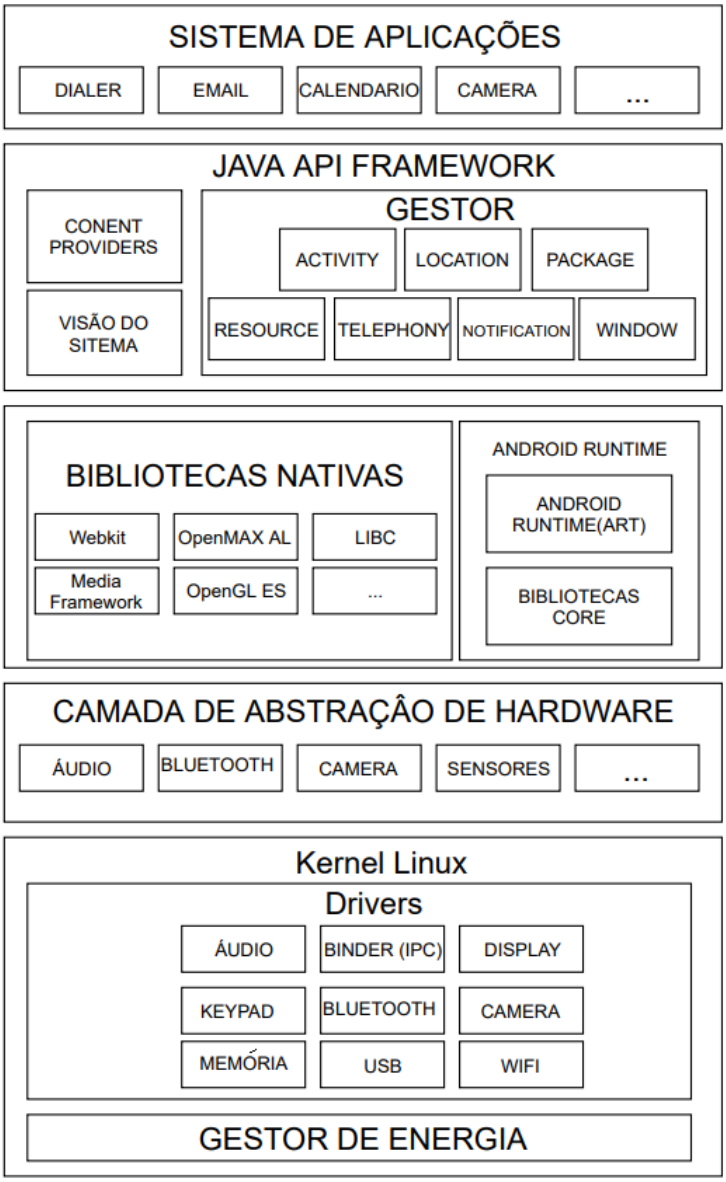


Figura 2.1: *Camadas do Software Android.*

2.1.1 Formato do ficheiro APK

O APK é a extensão de qualquer aplicação *Android*. Este permite ao dispositivo encontrar um ficheiro de instalação, assim como os programas com as extensões ".exe" ou ".msi" para Windows e Linux. Um APK pode ser aberto como um ficheiro comprimido (".zip" ou ".jar") e dentro deste é normal encontrarmos directórios ou ficheiros como:

- META-INF/: Contém o Manifest.xml, a assinatura da aplicação, a lista de recursos dos arquivos, entre outros;
- lib/: bibliotecas nativas para correr em arquitecturas de *Hardware* específicas (armeabi-v7a, x86, etc.);
- res/: recursos como imagens ou outros tipos de ficheiros que não faz parte do conteúdo que necessita de ser compilado;
- assets/: lista de recursos específicos para a aplicação;
- Manifest.xml: descreve o nome, a versão, e as permissões do APK.

Os directórios/ficheiros anteriormente mencionados definem o comportamento da aplicação e os recursos necessários [9] para a sua instalação e execução.

2.1.2 *Android Dalvik Virtual Machine* vs *Android Runtime*

A decisão de executar as aplicações dentro de pequenas máquinas virtuais foi algo sustentado por duas razoes fundamentais: a segurança e a independência das plataformas. Segurança porque, em teoria, o código em execução de uma aplicação fica totalmente isolado na máquina virtual, o que impede que qualquer aplicação tenha noção do seu verdadeiro ambiente de execução e que interaja directamente com outras aplicações, o que é ideal para evitar a propagação de Malware. Relativamente ao segundo pilar, com as máquinas virtuais, uma aplicação pode correr em qualquer arquitectura, seja ela ARM, MIPs ou x86 [39].

O sistema operativo *Android* apresenta duas soluções para criação das suas máquinas virtuais, o *Android Dalvik Virtual Machine* (Dalvik) e o *Android Runtime* (RTT). As aplicações RTT (*Android Runtime*) são caracterizadas da seguinte forma:

- Usam uma abordagem AOT (*Ahead-Of-Time*) [12], isto é, as aplicações são compiladas assim que são instaladas, o resultado é um menor consumo de memória e de processador.
- A *cache* da aplicação é criada quando o telefone é ligado, introduzindo algum atraso neste processo.

Já as aplicações Dalvik têm associadas as características:

- Concretizam uma abordagem *Just-in-Time*(JIT) [38], o que resulta num uso muito inferior de espaço em disco. A desvantagem é que as aplicações demoram mais tempo a iniciar.
- A *cache* das aplicações Dalvik está sempre a ser actualizada, o que torna os arranques do telefone mais rápidos.
- Tem melhor desempenho em casos de dispositivos com pouca memória interna porque só as aplicações em uso estão a ocupar espaço em memória.

O facto das máquinas virtuais Dalvik terem uma abordagem JIT e as máquinas virtuais RTT terem uma abordagem AOT, faz com que as segundas tenham maior desempenho [39].

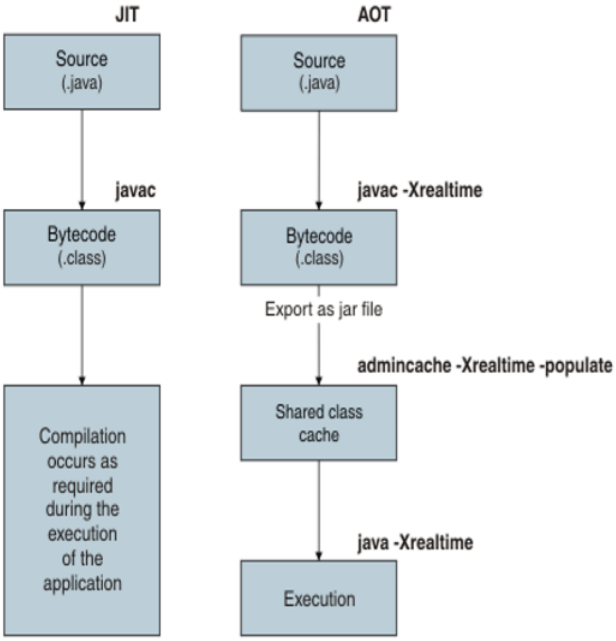


Figura 2.2: JIT vs AOT [39].

2.2 Vulnerabilidades *Android*

A comunidade OWASP define uma classificação para as aplicações móveis, assim como para as aplicações web. Nesta classificação iremos apenas contemplar as vulnerabilidades que têm como ponto de entrada uma aplicação *Android*, ou seja, o atacante para explorar o dispositivo da vítima tem de descobrir uma vulnerabilidade causada pelo código da aplicação instalada.

O *OWASP Mobile Top 10* é um estudo que nos descreve as vulnerabilidades com mais ocorrência nas aplicações *Android*. As últimas versões do último estudo revela as seguintes ameaças [29]:

- Uso impróprio da plataforma: Esta categoria foca-se na utilização imprópria de uma funcionalidade ou a falha no uso da plataforma de segurança. Esta categoria inclui *Intents*, as permissões, o uso inapropriado do *touchID*, do *keychain* ou de algum outro controlo de segurança que faz parte do *Android*;

- Armazenamento de dados inseguro: Armazenamento incorrecto de informação sensível, incluindo partilha de informação não intencional;

- Comunicação insegura: Limitações ao nível das comunicações, o que abrange as fracas configurações no *handshake* do protocolo TLS, o uso de versões do protocolo TLS vulneráveis, a fraca negociação de parâmetros e as transmissões em claro de informação sensível;

- Autenticação insegura: Gestão incorrecta da sessão de um utilizador ou identificação do utilizador;

- Criptografia insegura: Uso inapropriado dos algoritmos criptográficos, como por exemplo as cifras;

- Autorização insegura: Esta categoria foca-se nas falhas de autorização, contendo os erros de autenticação ou falha na realização de acções críticas numa aplicação;

- Qualidade de código: Fraca qualidade do código de uma aplicação que pode resultar em *buffer overflow*, *format strings* vulneráveis, ou outros tipos de erros que podem ser aproveitados para comprometer a execução correta das aplicações;

- Adulteração de código: Esta categoria foca-se na possibilidade de um atacante poder fazer engenharia inversa nas aplicações já instaladas, alterando o código para o seu proveito;

- Engenharia inversa: Capacidade do atacante a partir da aplicação, conseguir chegar ao código fonte, podendo desta forma ler os programas de forma legível para o seu proveito;

- Funcionalidades estranhas: Esta categoria foca-se em erros básicos que o programador pode deixar escapar, como por exemplo *passwords* em comentários no código da aplicação ou modo *debug* da aplicação activa.

2.3 *OWASP Mobile Security*

A comunidade OWASP tem dois projetos que se focam em apresentar soluções para aumentar a segurança das aplicações *Android*: o *OWASP Mobile Security Testing Guide* e o *OWASP Mobile Application Security Verification Standard*.

O *OWASP Mobile Security Testing Guide* é um livro que descreve como realizar testes de segurança e tarefas de engenharia inversa. [29]. Já o manual *OWASP Mobile Application Verification Standard* define uma *baseline* de segurança e requisitos de segurança para vários cenários de utilização. A OWASP resumiu, a partir destes dois estudos, as tarefas necessárias para avaliar um APK do ponto de vista de segurança. Uma vez que iremos automatizar a validação destas tarefas com o sistema ransAPK, vamos em seguida descrevê-las em maior detalhe.

Comecemos por apresentar as tarefas genéricas, que são explicadas através do preenchimento de uma folha de cálculo. Na Figura 2.3 podemos ver num *dashboard* os campos necessários a preencher para o início da avaliação de segurança. A *dashboard* tem a informação dividida em três grupos: os dados gerais do teste a realizar (parte superior da figura); a informação sobre a aplicação (bloco do meio); e informações sobre o proprietário da aplicação (bloco inferior) [26].

OWASP Mobile Application Security Checklist

Based on the OWASP Mobile Application Security Verification Standard

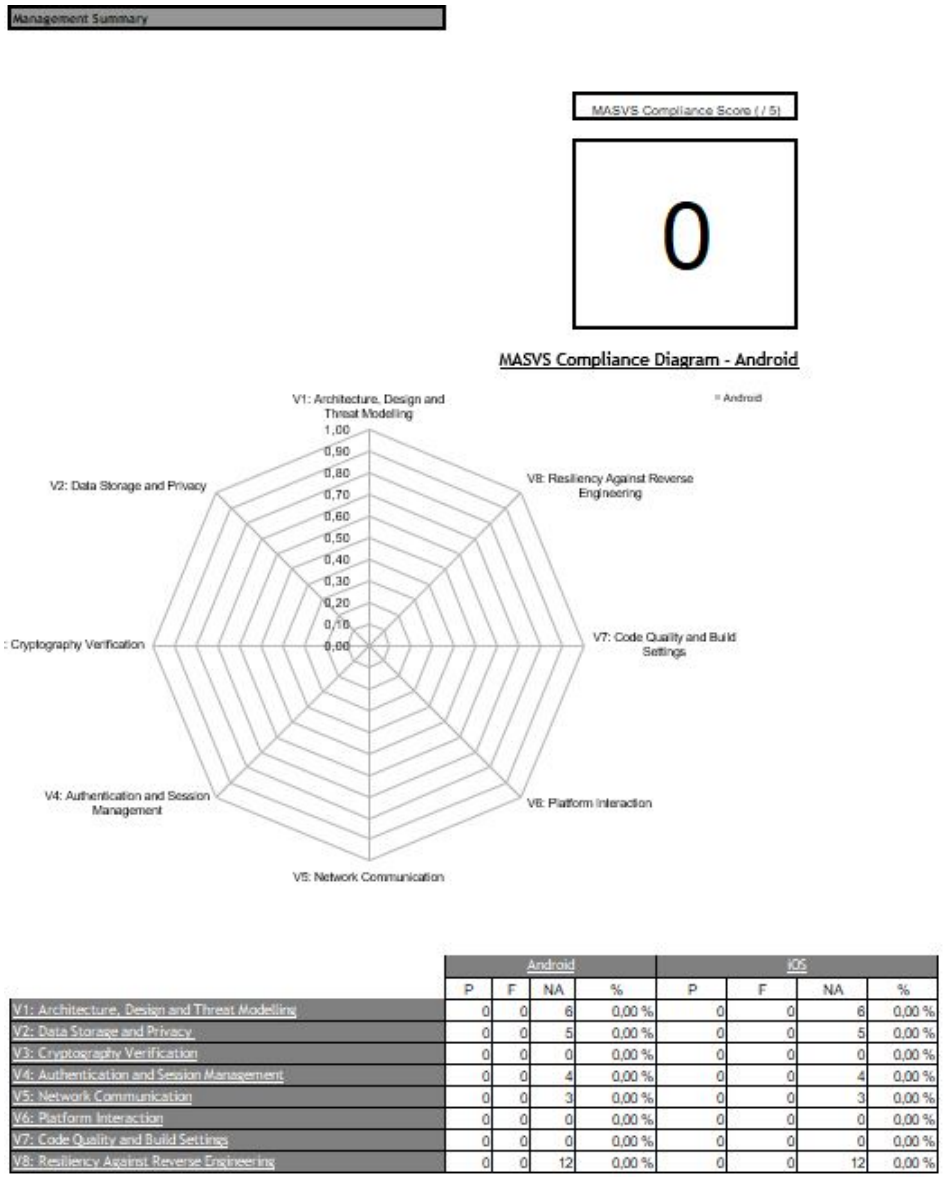
General Testing Information	
MASVS VERSION	1.1.4
Online version of the MASVS:	https://github.com/OWASP/owasp-masvs/blob/1.1.4/Document/
MSTG Version:	1.1.3
Online version of the MSTG:	https://github.com/OWASP/owasp-mstg/blob/1.1.3/Document/
The two rows above are used to construct the base for all hyperlinks in the Android and iOS cehcklists. Adjust to your specific use case to update all hyperlinks to a specific version of the MSTG	
Client Name:	
Test Location:	
Start Date:	
Closing Date:	
Name of Tester:	
Testing Scope	All available functions within the App <AppName>.
Verification Level	After consultation with <Customer> it was decided that only Level 1 requrirements are applicable to <AppName>.

Testing information Android	
Application Name:	
Google Play Store Link	
Filename	
Version	
SHA256 Hash of APK (Can be obtained by using shasum, openssl or sha256sum)	

Client Representatives and Contact Information	
Name:	
Org:	
Title:	
Phone:	
E-mail:	
Name:	
Org:	
Title:	
Phone:	
E-mail:	

Figura 2.3: *Dashboard*: dados da aplicação [29].

Em seguida temos o sumário da análise, onde uma folha de cálculo contém um gráfico que ilustra a maturidade do APK, auferido após concluída a avaliação (ver Figura 2.4). O gráfico é octogonal e cada vértice corresponde a um módulo específico de análise. A avaliação final de cada módulo tem um valor que pode ir de zero a um, com duas casas decimais. Por fim, quando todos os módulos estiverem preenchidos, é calculado um valor final que corresponde à maturidade de segurança na aplicação. Este valor está compreendido num intervalo de zero e cinco, sendo que zero significa que a aplicação chumba em todos os critérios de segurança e cinco quando a aplicação passa em todos os requisitos necessários.



	Android				iOS			
	P	F	NA	%	P	F	NA	%
V1: Architecture, Design and Threat Modelling	0	0	6	0,00 %	0	0	6	0,00 %
V2: Data Storage and Privacy	0	0	5	0,00 %	0	0	5	0,00 %
V3: Cryptography Verification	0	0	0	0,00 %	0	0	0	0,00 %
V4: Authentication and Session Management	0	0	4	0,00 %	0	0	4	0,00 %
V5: Network Communication	0	0	3	0,00 %	0	0	3	0,00 %
V6: Platform Interaction	0	0	0	0,00 %	0	0	0	0,00 %
V7: Code Quality and Build Settings	0	0	0	0,00 %	0	0	0	0,00 %
V8: Resiliency Against Reverse Engineering	0	0	12	0,00 %	0	0	12	0,00 %

Figura 2.4: *Resultado de cada módulo.* [29].

Os dois grupos seguintes de tarefas são requisitos de segurança *Android* e anti-engenharia inversa. Estas folhas são responsáveis por fornecer informação para o gráfico anteriormente descrito. São constituídos por 72 tópicos que precisam de ser preenchidos para o cálculo final da maturidade de segurança, estando estes divididos em L1, L2 e RE. Segundo a OWASP, nem todo o tipo de aplicações desenvolvidas necessitam do mesmo número de requisitos. Assim sendo, a comunidade descreve que existem três grupos de requisitos de segurança L1, L2 e RE.

Os tópicos L1 são considerados *Standard Security* e entendem-se como requisitos básicos que aumentam a qualidade do código, protegem a manipulação de dados confidenciais e a interacção com o sistema *Android*. Este nível é apropriado para todas as aplicações *Android*.

L2 - *Defense-in-Depth* que apresenta controlos avançados de segurança que vão para além dos requisitos definidos por L1. Para L2, deve existir um modelo de ameaça e a segurança deve ser parte integrante da arquitectura e desenho da aplicação. Este nível é apropriado para aplicações que manipulam dados confidenciais, como aplicações de transacções financeiras.

Por fim, existem os tópicos agrupados no RE - *Resiliency Against Reverse Engineering and Tampering*. Para passar neste nível, a aplicação deve ser resistente a ataques específicos, como a alteração de código ou a engenharia inversa. Neste tipo de aplicações é esperado que se use recursos de segurança do *Hardware* ou técnicas fortes de protecção de *Software*, bem como mecanismos de validação de protecção de *Software*. RE é aplicável a aplicações que tratam de dados sensíveis de diversos tipos, incluindo os relacionados com a propriedade intelectual ou a privacidade dos utilizadores. A figura 5.2 dá uma visão de como as três camadas são aplicadas a uma aplicação.

Resumidamente, as aplicações podem ser verificadas em L1 ou L2 com base na avaliação de risco prévia e no nível geral de segurança exigido. L1 é aplicável a todas as aplicações, enquanto L2 é geralmente recomendado para aplicações que lidam com dados e/ou funcionalidades mais sensíveis. O RE pode ser aplicado para verificar a resiliência contra ameaças específicas, ou extracção de dados confidenciais, além da verificação de segurança adequada [29].

Nas próximas secções iremos descrever em maior detalhe os principais tópicos tratados nos níveis L1, L2 e RE.

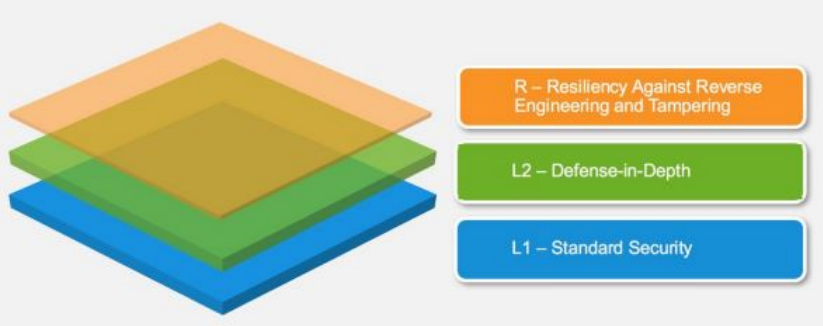


Figura 2.5: *Camadas de defesa de uma aplicação* [29].

2.3.1 Requisitos de arquitectura, desenho e modelação de ameaças

Estes requisitos referem-se à arquitectura e design de uma aplicação. Nesta secção são abordados tópicos como modelação de ameaças, SDLC (*Software Development Life Cycle*), gestão de chaves e utilizadores. Os requisitos que definem este módulo podem ser vistos na Tabela 2.1, bem como a camada que correspondem [29].

#	MSTG-ID	Descrição	L1	L2
1.1	MSTG-ARCH-1	Todos os componentes da aplicação estão identificados e são necessários.	X	X
1.2	MSTG-ARCH-2	Controlos de segurança nunca são forçados apenas no lado do cliente, mas também no respectivo ponto de acesso remoto.	X	X
1.3	MSTG-ARCH-3	Uma arquitectura de alto nível para a aplicação móvel e todos os serviços remotos foi definida e a segurança foi endereçada nessa arquitectura.	X	X
1.4	MSTG-ARCH-4	Informação considerada sensível no contexto da aplicação móvel está claramente identificada.	X	X
1.5	MSTG-ARCH-5	Todos os componentes da aplicação estão definidos em termos de funções negócio e/ou funções de segurança que os mesmos providenciam.		X
1.6	MSTG-ARCH-6	Modelação de ameaças para a aplicação móvel e os serviços remotos associados que identifica potenciais ameaças e contramedidas foi produzido.		X
1.7	MSTG-ARCH-7	Todos os controlos de segurança têm uma implementação centralizada.		X
1.8	MSTG-ARCH-8	Existe uma política explicita de como chaves de criptografia (caso existam) são geridas, e o ciclo de vida das chaves de criptografia é imposto. O ideal é seguir um padrão para gestão de chaves, como o NIST SP 800-57.		X
1.9	MSTG-ARCH-9	Existe um mecanismo para impor actualizações da aplicação móvel.		X
1.10	MSTG-ARCH-10	Segurança é endereçada em todas as partes do ciclo de vida do *Software*.		X
1.11	MSTG-ARCH-11	Uma política de divulgação responsável é usada e aplicada de forma efectiva.		X

Tabela 2.1: Requisitos de arquitectura, desenho e modelação de ameaças.

2.3.2 Armazenamento de dados e requisitos de privacidade

A protecção de dados confidenciais, como credenciais do utilizador e informação privada, é um foco importante na segurança de uma aplicação. Em primeiro lugar, dados confidenciais podem ser expostos involuntariamente a outras aplicações, quando em execução no mesmo dispositivo se os mecanismos do sistema operativo como o *inter-process communication* (IPC) forem usados incorrectamente. Os dados também podem de forma involuntariamente estar em armazenamentos *cloud*, *backups* ou *cache* do *keyboard*. Além disso, os telemóveis podem ser perdidos ou roubados mais facilmente em comparação com outros tipos de dispositivos.

No que diz respeito a armazenamento, existem protecções adicionais que podem ser implementadas para dificultar a recuperação dos dados confidenciais. Este módulo focase no telemóvel, mas não abrange políticas usadas pelo próprio telemóvel. Os requisitos que definem este módulo podem ser vistos na Tabela 2.2 os quais essencialmente correspondem ao nível de defesa de camada L2 [26]:

#	MSTG-ID	Descrição	L1	L2
2.1	MSTG-STORAGE-1	Os recursos de armazenamento de credenciais do sistema precisam ser usados para armazenar dados confidenciais, como PII, credenciais de utilizador ou chaves criptográficas.	X	X
2.2	MSTG-STORAGE-2	Nenhum dado confidencial deve ser armazenado fora do *container* da aplicação ou das instalações de armazenamento de credenciais do sistema.	X	X
2.3	MSTG-STORAGE-3	Nenhum dado confidencial é gravado nos *logs* da aplicação.	X	X
2.4	MSTG-STORAGE-4	Nenhum dado sensível é compartilhado com terceiros, a menos que seja uma parte necessária da arquitectura.	X	X
2.5	MSTG-STORAGE-5	A *cache* do teclado é desactivada em entradas de texto que processam dados confidenciais.	X	X
2.6	MSTG-STORAGE-6	Nenhum dado sensível é exposto por meio de mecanismos IPC.	X	X
2.7	MSTG-STORAGE-7	Nenhum dado sensível, como senhas ou *pins*, é exposto por meio da interface do utilizador.	X	X
2.8	MSTG-STORAGE-8	Nenhum dado sensível é incluído nos *backups* gerados pelo sistema operativo móvel.		X
2.9	MSTG-STORAGE-9	A aplicação remove dados confidenciais das visualizações quando movidos para o segundo plano.		X
2.10	MSTG-STORAGE-10	A aplicação não mantém dados confidenciais na memória por mais tempo do que o necessário, e a memória é limpa explicitamente após o uso.		X
2.11	MSTG-STORAGE-11	A aplicação impõe uma política mínima de segurança de acesso ao dispositivo, como exigir ao utilizador que defina uma senha para o dispositivo.		X
2.12	MSTG-STORAGE-12	A aplicação educa o utilizador sobre os tipos de informações de identificação pessoal processadas, bem como as melhores práticas de segurança que o utilizador deve seguir ao usar a aplicação.		X
2.13	MSTG-STORAGE-13	Nenhum dado sensível deve ser armazenado localmente no dispositivo móvel. Em vez disso, os dados devem ser recuperados através de um ponto de extremidade remoto quando necessário e apenas mantidos na memória.		X
2.14	MSTG-STORAGE-14	Se os dados confidenciais ainda precisarem ser armazenados localmente, eles deverão ser cifrados usando uma chave derivada do armazenamento com suporte de *Hardware* que requer autenticação.		X
2.15	MSTG-STORAGE-15	O armazenamento local da aplicação deve ser apagado após um número excessivo de tentativas de autenticação com falha.		X

Tabela 2.2: Armazenamento de dados e requisitos de privacidade.

2.3.3 Requisitos de criptografia

A criptografia é essencial quando se trata de proteger os dados armazenados numa aplicação. Esta é também uma categoria em que a abordagem definida pode falhar, especialmente quando as convenções padrões não são seguidas. Os requisitos que definem esta categoria podem ser vistos na Tabela 2.3. De salientar que todos eles pertencem a L1 e L2 para a criptografia ser considerada um dos pilares de implementação de segurança.

#	MSTG-ID	Descrição	L1	L2
3.1	MSTG-CRYPTO-1	A aplicação não depende de criptografia simétrica com chaves codificadas como único método de criptografia.	X	X
3.2	MSTG-CRYPTO-2	A aplicação usa implementações seguras das primitivas criptográficas.	X	X
3.3	MSTG-CRYPTO-3	A aplicação usa primitivas criptográficos apropriadas para o caso de uso específico, configuradas com parâmetros que seguem as práticas recomendadas no sector.	X	X
3.4	MSTG-CRYPTO-4	A aplicação não usa protocolos criptográficos ou algoritmos considerados obsoletos para fins de segurança.	X	X
3.5	MSTG-CRYPTO-5	A aplicação não reutiliza a mesma chave criptográfica para vários fins.	X	X
3.6	MSTG-CRYPTO-6	Todos os valores aleatórios são criados usando um gerador de números aleatórios suficientemente seguro.	X	X

Tabela 2.3: Requisitos de criptografia.

2.3.4 Requisitos de autenticação e gestão de sessão

Na maioria dos casos, o *login* de utilizadores num serviço remoto é parte integrante da arquitectura geral de uma aplicação. Embora a maior parte da lógica ocorra no servidor, este módulo define alguns requisitos básicos sobre a gestão de contas e sessões de utilizador [29]. Os requisitos que definem este módulo podem ser vistos na Tabela 2.4 [29]. Todos eles pertencem a L2 enquanto metade são requisitos de L1.

#	MSTG-ID	Descrição	L1	L2
4.1	MSTG-AUTH-1	Se a aplicação fornece aos utilizadores acesso a um serviço remoto, alguma forma de autenticação, como autenticação de nome de utilizador/senha, é realizada no servidor remoto.	X	X
4.2	MSTG-AUTH-2	Se a gestão de sessão com monitorização de estado for usado, o terminal no servidor usará identificadores de sessão gerados aleatoriamente para autenticar as solicitações do cliente sem enviar as credenciais do utilizador.	X	X
4.3	MSTG-AUTH-3	Se a autenticação baseada em *token* sem estado for usada, o servidor fornecerá um *token* que foi assinado usando um algoritmo seguro.	X	X
4.4	MSTG-AUTH-4	A sessão existente quando o utilizador efectua *logout*, é também feito *logout* no servidor remoto.	X	X
4.5	MSTG-AUTH-5	Uma política de *password* existe e é aplicada no ponto de servidor remoto	X	X
4.6	MSTG-AUTH-6	O servidor remoto implementa um mecanismo de protecção contra o envio de credenciais num número excessivo de vezes.	X	X
4.7	MSTG-AUTH-7	As sessões são invalidadas no servidor remoto após um período predefinido de inactividade e os *tokens* de acesso expiraram.	X	X
4.8	MSTG-AUTH-8	A autenticação biométrica, se houver, não é associada a eventos (ou seja, usada numa API que simplesmente retorna ”verdadeiro”ou ”falso”). Em vez disso, é baseado no desbloqueio do *keychain/keystore*.		X
4.9	MSTG-AUTH-9	Um segundo factor de autenticação existe no servidor remoto e o requisito 2FA é aplicado de forma consistente.		X
4.10	MSTG-AUTH-10	As transacções sensíveis requerem autenticações progressivas.		X
4.11	MSTG-AUTH-11	A aplicação informa o utilizador sobre todas as actividades confidenciais de sua conta. Os utilizadores podem visualizar uma lista de dispositivos, visualizar informações contextuais (endereço IP, localização, etc.) e bloquear dispositivos específicos.		X
4.12	MSTG-AUTH-12	Os modelos de autorização devem ser definidos e aplicados no terminal remoto.		X

Tabela 2.4: Requisitos de autenticação e gestão de sessão.

2.3.5 Requisitos de comunicação em rede

O objectivo dos requisitos listados nesta secção é assegurar a confidencialidade e a integridade da informação trocada entre a aplicação e os servidores que a suportam. No mínimo, uma aplicação deve configurar, usando o protocolo TLS com as configurações apropriadas, um canal seguro para a comunicação em rede. Os requisitos para garantir uma comunicação segura, podem ser vistos na Tabela 2.5 [29] estando todas elas em L2.

#	MSTG-ID	Descrição	L1	L2
5.1	MSTG-NETWORK-1	Os dados são cifrados na rede usando TLS. Este canal seguro é usado consistentemente em toda a aplicação.	X	X
5.2	MSTG-NETWORK-2	As configurações de TLS estão de acordo com as melhores práticas actuais, ou o mais próximo possível, caso o sistema operativo do dispositivo não suporte os padrões recomendados.	X	X
5.3	MSTG-NETWORK-3	A aplicação verifica o certificado X.509 do ponto de acesso remoto, quando o canal seguro é estabelecido. Apenas certificados assinados por uma autoridade certificadora de confiança são aceites.	X	X
5.4	MSTG-NETWORK-4	A aplicação usa a sua própria autoridade de certificação, ou então recorre a *certificate pining* na chave pública ou *certificate pining* no certificado do servidor, e subsequentemente não aceita ligações a pontos de acesso que ofereçam um certificado ou chave diferentes, mesmo que assinados por uma autoridade certificadora de confiança.		X
5.5	MSTG-NETWORK-5	A aplicação não depende apenas de um único canal de comunicação inseguro (e-mail ou SMS) para operações críticas, tais como inscrições e recuperação de contas.		X
5.6	MSTG-NETWORK-6	A aplicação depende apenas de ligações actualizada e bibliotecas seguras.		X

Tabela 2.5: Requisitos de comunicação em rede.

2.3.6 Requisitos de interacção da plataforma

Este grupo de requisitos garantem que uma aplicação usa APIs da plataforma e componentes padrão de maneira segura. Além disso, os requisitos abrangem a comunicação entre aplicações, com o uso de IPC[29]. Os requisitos que definem este módulo podem ser vistos na Tabela 2.6, onde novamente todos os requisitos pertencem a L2 e 75% a L1.

#	MSTG-ID	Descrição	L1	L2
6.1	MSTG-PLATFORM-1	A aplicação solicita apenas o conjunto mínimo de permissões necessárias.	X	X
6.2	MSTG-PLATFORM-2	Todos as entradas de fontes externas e do utilizador são validadas. Isso inclui dados recebidos por meio da interface do utilizador, mecanismos de IPC, como *intents*, URLs personalizados e fontes de rede.	X	X
6.3	MSTG-PLATFORM-3	A aplicação não exporta funcionalidades confidenciais por meio de esquemas de URL personalizados, a menos que esses mecanismos estejam devidamente protegidos.	X	X
6.4	MSTG-PLATFORM-4	A aplicação não exporta funcionalidades confidenciais por meio dos recursos do IPC, a menos que esses mecanismos sejam protegidos de forma adequada.	X	X
6.5	MSTG-PLATFORM-5	O JavaScript está desactivado em *WebViews*, a menos que seja explicitamente necessário.	X	X
6.6	MSTG-PLATFORM-6	Os *WebViews* são configurados para permitir apenas o conjunto mínimo de manipuladores de protocolo necessários (de preferência, apenas HTTPS é compatível). Manipuladores potencialmente perigosos, como *file*, *tel* e *app-id*, estão desactivados.	X	X
6.7	MSTG-PLATFORM-7	Se os métodos nativos da aplicação forem expostos num *WebView*, é necessário verificar se a *WebView* executa o JavaScript contido no pacote da aplicação.	X	X
6.8	MSTG-PLATFORM-8	A desserialização de objectos, se houver, é implementada usando APIs de serialização segura.	X	X
6.9	MSTG-PLATFORM-9	A aplicação protege-se contra ataques de sobreposição de ecrã.		X
6.10	MSTG-PLATFORM-10	A *cache*, o armazenamento e os recursos carregados de uma *WebView* (JavaScript etc.) devem ser limpos antes de a *WebView* ser destruída.		X
6.11	MSTG-PLATFORM-11	Evitar o uso de teclados personalizados de terceiros sempre que dados confidenciais sejam inseridos.		X

Tabela 2.6: Requisitos de interacção da plataforma.

2.3.7 Qualidade de código e requisitos de configuração de construção

O objectivo destes requisitos é garantir que as práticas básicas de desenvolvimento de código seguro são seguidas na construção da aplicação, e que os recursos de segurança já existentes na plataforma *Android* são activados. Um sumário dos requisitos tratados neste grupo é apresentado na Tabela 2.7 [29]. É visível que todos os requisitos estão presentes em L1 e L2, denotando que a implementação do código seguro é a base para o desenvolvimento de aplicações seguras e por conseguinte o cumprimento de inúmeros requisitos apresentados nas tabelas anteriores.

#	MSTG-ID	Descrição	L1	L2
7.1	MSTG-CODE-1	A aplicação é assinada e associada a um certificado válido, a chave privada encontra-se devidamente protegida.	X	X
7.2	MSTG-CODE-2	A aplicação é desenvolvida em modo de distribuição, com configurações adequadas para a implementação de uma nova versão de produção (ex. não depurável).	X	X
7.3	MSTG-CODE-3	Os símbolos de depuração são removidos dos binários nativos.	X	X
7.4	MSTG-CODE-4	O Código de depuração e o código de assistência ao programador (ex. código de teste, conteúdos secretos e configurações ocultas) são removidos. A aplicação não regista mensagens de erro comuns ou de depuração.	X	X
7.5	MSTG-CODE-5	Todos os componentes de terceiros usados pela aplicação, como bibliotecas e frameworks, são identificados e adequados para a detecção de vulnerabilidades específicas.	X	X
7.6	MSTG-CODE-6	A aplicação identifica e trata possíveis excepções.	X	X
7.7	MSTG-CODE-7	A lógica de tratamento de erros em controlos de segurança, nega o acesso por omissão.	X	X
7.8	MSTG-CODE-8	A memória é alocada, libertada e usada de forma segura pela aplicação.	X	X
7.9	MSTG-CODE-9	Recursos de segurança oferecidos pelo conjunto de ferramentas de programação, como minimização de bytes de código, protecção da pilha (*stack*), suporte ao *Position-independent code* (PIE) e a contagem automática de referências, são activados.	X	X

Tabela 2.7: Qualidade de código e requisitos de configuração de construção.

2.3.8 Requisitos de resiliência

Os requisitos deste grupo devem ser aplicados conforme necessário, com base na avaliação dos riscos causados por adulteração não autorizada da aplicação e/ou engenharia reversa do código [29]. Os requisitos que definem este módulo podem ser vistos na Tabela 2.8. Este grupo de requisitos é o que define a camada de defesa RE, logo todos eles pertencem a esta camada.

#	MSTG-ID	Descrição	RE
8.1	MSTG-RESILIENCE-1	A aplicação detecta e responde à presença de um dispositivo a executar-se em modo *root*, alertando o utilizador para fechar a aplicação.	X
8.2	MSTG-RESILIENCE-2	A aplicação previne a depuração, detecta e responde a um depurador ligado à aplicação. Todos os protocolos de depuração devem ser tidos em conta.	X
8.3	MSTG-RESILIENCE-3	A aplicação detecta e responde à manipulação de ficheiros executáveis e informação crítica dentro da sua *sandbox*.	X
8.4	MSTG-RESILIENCE-4	A aplicação detecta e responde à presença de ferramentas ou estruturas de engenharia reversa no dispositivo.	X
8.5	MSTG-RESILIENCE-5	A aplicação detecta e responde se executada num emulador.	X
8.6	MSTG-RESILIENCE-6	A aplicação detecta e responde à manipulação de código e informação no seu próprio espaço de memória.	X
8.7	MSTG-RESILIENCE-7	A aplicação implementa mais que um mecanismo na categoria 8.1 a 8.6. É de notar que a resiliência aumenta com o número, diversidade e originalidade dos sistemas usados.	X
8.8	MSTG-RESILIENCE-8	Os mecanismos de detecção desencadeiam respostas de diferentes tipos, incluindo respostas atrasadas e furtivas.	X
8.9	MSTG-RESILIENCE-9	A ofuscação é aplicada a defesas programáticas, que por sua vez impedem a desofuscação por meio de análise dinâmica.	X
8.10	MSTG-RESILIENCE-10	A aplicação implementa uma funcionalidade de *DEVICE BINDING*, usando uma impressão digital do dispositivo, derivada de várias propriedades únicas do mesmo.	X
8.11	MSTG-RESILIENCE-11	Todos os ficheiros executáveis e bibliotecas pertencentes à aplicação são cifrados ao nível do ficheiro, e / ou o código importante ou segmentos de informação dentro dos executáveis são cifrados ou comprimidos. Análise estática básica não revela o código ou a informação importante.	X
8.12	MSTG-RESILIENCE-12	Se o objectivo da ofuscação é proteger contra roubo de código, um esquema de ofuscação deve ser usado de forma robusta contra métodos de desofuscação manuais e automáticos, considerando as investigações publicadas. A efectividade do esquema de ofuscação deve ser verificada com testes manuais. É de notar que as características de isolamento baseado em *Hardware* são preferidas em relação a uma ofuscação sempre que possível.	X
8.13	MSTG-RESILIENCE-13	Como uma defesa em profundidade, além de ter um reforço sólido das partes comunicantes, a criptografia do dispositivo pode ser aplicada para impedir ainda mais a espionagem.	X

Tabela 2.8: Requisitos de resiliência

2.4 Engenharia reversa para análise estática de código

No *Android*, é geralmente inconveniente executar uma análise estática directamente no Assembly da aplicação. Neste caso, a aplicação está escrita numa linguagem de baixo nível, o que dificulta a compreensão do seu código. Logo, o uso de técnicas de engenharia inversa foram adaptadas para permitir a análise de programas, fazendo a tradução do código da aplicação para outra linguagem de alto nível. No *Android*, as linguagens intermédias podem ser, por exemplo o Smali [41], o Jasmin [24] e o Jimple [3]. Estas foram criadas para representar o *Bytecode* executável do Dalvik de forma mais legível. Existem três ferramentas, APKtool [1], Dex2Jar [5] e Soot [36], que fazem a transformação de Java *Bytecode* em Smali, Jasmin e Jimple, respectivamente. Uma das preocupações que se deve ter em conta quando se escolhe uma ferramenta de tradução é a sua precisão. A imprecisão da tradução coloca toda a análise em causa, devido às incoerências na tradução do verdadeiro código.

Neste momento, ainda não se conhece qual a melhor ferramenta capaz de executar com maior precisão uma tradução a partir do Assembly, assegurando que as aplicações *Android* traduzidas mantenham o seu comportamento original [2]. Existe um estudo baseado em eventos estatísticos que avalia a precisão da tradução de uma linguagem de alto nível do *Android* para uma linguagem intermédia por meio de uma validação automatizada de comportamentos do programa. Neste estudo, na abordagem utilizada foi concebido um sistema que usa a ferramenta Monkey, a qual personifica um utilizador ao gerar automaticamente eventos na interface de uma aplicação. Foi avaliado o desempenho em aplicações *Android*, obtidas de 26 categorias distintas do Google Play. Foi descoberto que o APKtool realiza a tradução mais precisa, pois as aplicações executadas com a tradução em Smali preservam a maioria dos comportamentos originais.

2.5 Análise de código dinâmica

É comum ter como primeira abordagem de análise de uma aplicação *Android* técnicas de análise estática ou dinâmica. A maioria das ferramentas tradicionais adoptam técnicas de análise estática devido ao seu baixo custo e propriedades de alto desempenho. No entanto, as aplicações podem ser ofuscadas e é comum trabalhar com técnicas de análise dinâmica para obtenção de informações da aplicação, os quais permitirão fornecer um comportamento profundo da mesma. Embora já existam muitas ferramentas baseadas em técnicas de análise dinâmica, a capacidade destas ferramentas é desconhecida [17].

Seria fácil entender a capacidade de uma ferramenta de análise dinâmica mediante a cobertura de código. No entanto, até onde se sabe, não existe uma abordagem universal para medir a cobertura de código efectuada por todas as ferramentas de análise dinâmica, especialmente quando uma ferramenta é apenas acedida remotamente. No artigo, Code

Coverage Measurement for *Android Dynamic Analysis Tools*, foi proposta uma abordagem para medir a cobertura de código para ferramentas de análise dinâmica *on-line* e *off-line*. Em seguida, foram escolhidas ferramentas *on-line*, incluindo ABM, Anubis, CopperDroid, Tracedroid, bem como ferramentas *off-line*, incluindo emulador *Android* padrão, DroidBox e DroidScope. Os resultados de medição mostram que a cobertura média é de 20% a 60% para as ferramentas. Acredita-se que a abordagem pode fornecer mais informações para investigadores e programadores de modo a entender e melhorar a capacidade das técnicas de análise dinâmica.

2.6 Ferramentas de análise autónoma

Existem algumas ferramentas com a capacidade de fazer uma análise autónoma de diversos requisitos de segurança, produzindo um relatório final que sumariza o observado. A maior parte destas soluções são de código fechado e algumas são pagas, o que dificulta a compreensão exacta das suas capacidades e maneira como operam. De forma a percebermos o potencial destas soluções, apresentamos de forma resumida as suas principiais características.

Começando pela aplicação ImmuniWeb [19], que oferece uma combinação da análise estática e dinâmica para aplicações móveis. Esta aplicação abrange de forma ampla os critérios elencados no *Mobile OWASP Top 10* [28] , no SANS Top 25 [33] e no PCI DSS 6.5.1-10 [34] para o seu código. A aplicação é de código fechado e paga, e em suma apresenta as seguintes capacidades:

- Conformidade com PCI DSS e GDPR [21] [4].
- Pontuações CVE, CWE e CVSSv3.
- Directrizes de correcção accionáveis.
- Integração de ferramentas SDLC e CI / CD [20].
- *Patching* virtual com um clique via WAF.
- Acesso 24/7 para analistas de segurança.

Outra ferramenta é o Codified Security [4], uma ferramenta popular em testes de segurança de aplicações móveis. A ferramenta identifica e corrige as vulnerabilidades de segurança, garantindo que a aplicação é segura. Esta ferramenta segue uma abordagem programática para testes de segurança, o que garante que os resultados são escaláveis. As suas características principais são:

- Realiza os testes de forma automatizada, detectando falhas de segurança no código da aplicação *Android* e fornecendo *feedback* em tempo real.

- Recorre à aprendizagem máquina e à análise estática de código.
- Oferece suporte para testes estáticos e dinâmicos de segurança de aplicações móveis.
- Produz relatórios que ajudam a resolver os problemas no código do lado do cliente.

O Mobile Security Framework (MobSF) [27] é uma *framework* de testes de segurança automatizado para plataformas *Android*, iOS e Windows. Esta executa análise estática e dinâmica para os testes de segurança de aplicações móveis. As características principais são:

- É uma ferramenta de código-aberto para testes de segurança em aplicações móveis.
- O ambiente de teste pode ser facilmente configurado, podendo ser instalador num ambiente local.
- Suporta código-fonte binário e compactado.
- Suporta testes de segurança de API da Web usando API Fuzzer.

Para começarmos a usar o MobSF basta fazermos o *download* do seu repositório, (actualmente encontra-se no GitHub), executar os comandos de instalação e iniciar o seu servidor local. Após iniciada, teremos acesso à página da ferramenta que nos permite carregar um ficheiro do tipo APK. A análise do ficheiro começa assim que o MobSF detecta que foi carregado um ficheiro, bastando aguardar para que o relatório correspondente à aplicação carregada seja disponibilizado. Uma vez que a aplicação é de código-aberto, podemos ver quais os mecanismos que esta utiliza para realizar o processo anteriormente descrito. Para a conversão do Java *Bytecode* em código Java são usadas duas ferramentas, a primeira APKTool serve para traduzir o ficheiro ".Dex" em código Smali, e de seguida usa a ferramenta Dex2Jar para converter os ficheiro Smali em classes Java. Por fim, analisa todos os ficheiros Java de forma estática com a biblioteca Libsast. Esta biblioteca é open-source e foi criada pelo mesmo autor da ferramenta MobSF.

Capítulo 3

Arquitectura do sistema ransAPK

Neste capítulo será apresentada a arquitectura e os principais componentes do sistema proposto e desenvolvido, o ransAPK. Este sistema tem por objectivo facilitar as tarefas de um analista, ou de um programador, ao inspeccionar uma aplicação *Android* sob o ponto de vista de segurança.

O ransAPK é um sistema com capacidade para identificar se foram implementados os requisitos de segurança suficientes e necessários para proteger uma aplicação contra uma actividade ilícita. Neste sentido, o ransAPK segue as recomendações da *OWASP - Mobile Security Project*, e verifica de forma automatizada os requisitos definidos pela mesma. O capítulo, numa primeira instância, apresenta quais as necessidades que o sistema a desenvolver deverá obedecer, ou seja, o que o ransAPK deve satisfazer. Seguidamente, descrevem-se os requisitos de segurança que o sistema proposto verifica, assinalando quais deles foram automatizados. Por fim, explica-se a visão geral da arquitectura do ransAPK, ilustrando o fluxo para optimizar os requisitos e o cálculo de maturidade de uma aplicação.

A abordagem seguida por este sistema baseia-se no uso da análise dinâmica e estática, que é introduzida pelos módulos incluídos nas diferentes fases da execução. Este é o comportamento fundamental do *back-end*, mas para seguir a metodologia definida pela comunidade OWASP na íntegra, o ransAPK contém também uma interface de interacção com o utilizador para o preenchimento manual dos restantes requisitos. O ransAPK não é só um sistema de automatização, mas também um guia de implementação. A interface disponibiliza ao utilizador informação de como analisar um requisito em concreto e quais as validações necessárias para satisfazer os requisitos que se encontram em falta.

3.1 Requisitos de sistema

Qualquer sistema de *software* deve cumprir um conjunto de características, sendo por norma definido através de requisitos de sistema.

Genericamente, no caso do ransAPK, as necessidades estão relacionadas com a

automatização do processo de validação dos requisitos OWASP e com a simplicidade de interacção com o utilizador. Deste modo, o sistema deve cumprir com as seguintes condições:

- **Modularidade -** Os componentes do sistema devem ser modulares, havendo assim uma independência entre eles. Desta forma, é possível alterar e perceber com maior facilidade a implementação de cada um dos componentes.
- **Versatilidade -** Adição de novos tipos de análise não deve implicar mudanças significativas no código fonte, nomeadamente em módulos não directamente relacionados com a tarefa.
- **Usabilidade -** As configurações necessárias para colocar o sistema em funcionamento devem ser simples de entender, com a documentação correcta.
- **Compatibilidade -** O sistema deve ser compatível com qualquer sistema operativo.
- **Desempenho -** Os resultados produzidos pelo sistema devem ser coerentes e o sistema deve minimizar os falsos positivos.
- **Eficiência -** O sistema deve realizar apenas as tarefas necessárias para produzir o melhor resultado possível, cumprindo os restantes requisitos, e sem atrasos desnecessários.

3.2 Requisitos de segurança alvo

Esta secção apresenta os 34 requisitos de segurança que o ransAPK avalia, os quais se encontram subdivididos nas oito categorias definidas pela comunidade OWASP (presentes na Secção 2.3). Para cada requisito é também descrita a abordagem proposta para a sua validação automatizada. Para tal, o ransAPK utiliza técnicas de análise estática e dinâmica de modo a obter uma resposta. Na selecção dos requisitos a automatizar foi tido em conta a necessidade de abranger o maior número de grupos definidos pela comunidade OWASP. O ransAPK apresenta uma técnica de validação para pelo menos um requisito de cada grupo.

Requisitos de arquitectura, desenho e modelação de ameaças. É um conjunto que define a estrutura da aplicação a desenvolver e o seu ciclo de desenvolvimento. Apenas o requisito 1.9, pertencente ao L2, tem a possibilidade de ser automatizado. Este requisito impõe que uma aplicação, sempre que exista uma actualização, obrigue o utilizador a aplicá-la e o impeça de continuar a usar a versão actual. A sua validação passa pela realização de uma análise estática sobre o código Java. Para verificar qual a versão actual da aplicação o programador terá de fazer chamadas a código de bibliotecas específicas. A partir da API de nível 21 (*Android* 5.0), junto com a *Play Core Library*, as aplicações

podem ser forçadas a serem actualizadas. Este mecanismo é baseado no uso da biblioteca *AppUpdateManager*. Existe também a alternativa do uso do *Firebase*, que é uma plataforma desenvolvida pela Google, para a criação de aplicações *Android*. Nem todas as actualizações são de segurança, mas quando o são, o facto de estas serem atrasadas, pode deixar o dispositivo vulnerável a ataques. A regra criada pelo sistema ransAPK verifica se existem usos da biblioteca *AppUpdateManager*, e caso existam, será testado se os métodos necessários para validar a versão actual estão a ser chamados.

Armazenamento de dados e requisitos de privacidade. Proteger *tokens* de autenticação, informações privadas e outros dados confidenciais, é o foco deste módulo. Os dados públicos devem estar disponíveis para todos, mas os dados confidenciais e privados devem ser protegidos. Para além dos dados confidenciais, é preciso garantir que os dados lidos de qualquer fonte de armazenamento são validados e possivelmente limpos. A validação geralmente não vai além de assegurar que os dados apresentados são do tipo solicitado. Com o uso de mecanismos de segurança adicionais, como por exemplo o HMAC [15], pode-se também assegurar a integridade dos dados. O ransAPK usa a análise estática para a verificação de seis requisitos deste grupo.

As regras criadas procuram validar se quando um programador gera um ficheiro, atribui as permissões correctas, ou seja, se a aplicação é a única capaz de ler o ficheiro criado. Se houver necessidade de escrever para um ficheiro, este é feito com a sanitização das variáveis. Já sobre o conteúdo do ficheiro, o ransAPK verifica se os ficheiros criados pela aplicação contêm *passwords* ou nomes de utilizador. No entanto, este conjunto de requisitos não se foca apenas na criação de ficheiros e no seu conteúdo. Existem duas regras extra. Uma dessas regras procura variáveis sensíveis, ou seja, que possam guardar *passwords* de utilizadores ou de sistema. A outra regra verifica se as acções do sistema estão a ser escritas no terminal. Por fim, ainda sobre este conjunto, existem acções do utilizador que podem colocar os seus dados sensíveis fora da zona de segurança criada pela app. Para isso o ransAPK verifica se existem funções que previnem que o utilizador copie texto da aplicação, que o utilizador faça a captura da tela, ou na necessidade de usar o teclado, que este não apresenta sugestões em *cache* para os campos que necessitam de preenchimento manual. As regras para este conjunto procuram o uso das funções necessárias para a execução das operações que foram anteriormente descritas.

Requisitos de criptografia. Este conjunto de requisitos desempenham um papel especialmente importante na protecção dos dados do utilizador, principalmente quando se está num ambiente de dispositivos *Android*, onde o acesso físico ao dispositivo pelos atacantes é um cenário provável. Este grupo foca-se na validação de aspectos criptográficos e nas práticas mais relevantes para aplicações *Android*. Estas práticas são recomendadas e validadas independentemente do sistema operativo do dispositivo. Neste conjunto o ransAPK procura validar como é que são instanciadas as funções de cifra ou os algoritmos de síntese. São também gerados alertas caso existam instâncias de algoritmos vulneráveis

ou de vectores de inicialização fracos como, por exemplo, vectores compostos por oito *bytes* de zeros ou oito *bytes* de sequências crescentes.

Requisitos de autenticação e gestão de sessão. Uma aplicação autentica o utilizador em relação às credenciais armazenadas localmente no dispositivo. Por outras palavras, o utilizador "desbloqueia" o seu dispositivo (ou alguma camada interna de funcionalidades) ao fornecer um *pin*, senha ou impressão digital válida. Esta é verificada por meio de referência a dados locais. Por norma este processo é invocado para garantir que funcionalidades críticas sejam activadas apenas pelo proprietário do dispositivo. Para este conjunto de requisitos foram criadas duas regras, uma que permite verificar se existem validações locais de *password* e outra que verifica se a aplicação contém as permissões para o uso de impressão digital.

Requisitos de comunicação em rede. Praticamente todas as aplicações com permissões para acesso à *Internet* usam o protocolo HTTP ou HTTP sobre uma camada de transporte segura (TLS), para enviar e receber dados de servidores remotos. Este módulo foca-se apenas na verificação da ligação entre o telemóvel e o servidor, validando se esta emprega a melhor configuração possível para evitar ataques de homem-no-meio [30]. Para esta validação, são avaliados quais os algoritmos de cifra que estão configurados no TLS entre o dispositivo do utilizador e o servidor remoto. Também é verificado no código da aplicação se existem bibliotecas para *ssl pinning*, e se as configurações de SSL usadas nas *WebViews* locais estão correctamente configuradas.

Requisitos de interacção da plataforma. O *Android* atribui uma identidade de sistema distinta (ID de utilizador e ID de grupo do Linux) para cada aplicação instalada. Como cada aplicação *Android* opera numa *sandbox*, para que as aplicações possam aceder a dados fora da sua *sandbox* têm de pedir uma permissão, que é validada pelo utilizador do dispositivo. Este conjunto de requisitos verifica se as configurações de segurança, como as permissões de uma aplicação *Android*, se encontram correctamente configuradas para que um utilizador não obtenha acesso a zonas que possam comprometer informação segura/confidencial ou escalar privilégios. O ransAPK para os *Requisitos de interacção da plataforma* valida o uso de bibliotecas vulneráveis como *libraryDeserialization* que podem originar execução arbitrária de código, ou se existem configurações que possibilitam a execução de JavaScript.

Qualidade de código e requisitos de configuração de construção. O sistema operativo *Android* requer que todos os APKs sejam assinados digitalmente antes de serem instalados ou executados. A assinatura digital é usada para verificar a identidade do proprietário. Quando um APK é assinado, é-lhe anexado um certificado de chave pública. Este certificado associa exclusivamente o APK à organização que o desenvolveu e à sua chave pública e privada. Estes requisitos começam por confirmar se o pacote oficial de uma aplicação foi assinado com os esquemas v1, v2 e v3. Esta configuração também deve ter todos os valores de *debug* a falso. Estas duas configurações são detectadas por regras

do sistema ransAPK. Ainda sobre este conjunto de requisitos, é descrito pela OWASP que não devem existir IPs no código sem estarem ofuscados. O ransAPK usa uma regra com uma expressão regular para detectar este requisito.

Requisitos de resiliência. Este conjunto de requisitos servem para proteger a propriedade intelectual do dono da aplicação e para proteger o normal funcionamento de uma aplicação. Para que os ataques não aconteçam é sugerido a quem desenvolve a aplicação que ofusque o código e que detecte mecanismos que possam colocar em causa o normal funcionamento da aplicação. Ou seja, a aplicação deve detectar se o dispositivo está *rooted*, se existem processos em execução já conhecidos com a capacidade de alterar valores em memória e se a aplicação está a ser executada num dispositivo físico ou emulado. O ransAPK neste conjunto de requisitos executa em primeiro lugar regras para validar se existem bibliotecas que detectem requisitos deste conjunto, como por exemplo a *DexGuard*, a *superUser* ou a *SafetyNetApi*. Em segundo lugar, instala a aplicação num dispositivo físico e tenta retirar o código em execução.

Conforme descrito anteriormente, o ransAPK tem a capacidade de validar automaticamente os requisitos especificados pela OWASP e um sumário da abordagem para a sua análise pode ser vista nas Tabelas 3.1 e 3.2.

Requisitos de arquitectura, desenho e modelação de ameaças		
#	MSTG-ID	Abordagem de automatização
1.9	MSTG-ARCH-9	Valida se a aplicação faz uso da biblioteca *AppUpdateManager* e chama as funções necessárias para verificar a versão actual da aplicação.
Armazenamento de dados e requisitos de privacidade		
#	MSTG-ID	Abordagem de automatização
2.2	MSTG-STORAGE-2	Procura a criação de ficheiros temporários ou ficheiros que estejam em modo escrita e/ou leitura para todos os utilizadores.
2.5	MSTG-STORAGE-5	Verifica se o teclado usado sugere palavras baseadas nas suas utilizações.
2.7	MSTG-STORAGE-7	Valida se a aplicação confia em certificados auto assinados ou certificados que não são assinados por entidades certificadoras.
2.9	MSTG-STORAGE-9	Valida se aplicação está com o modo *debug* activo.
2.10	MSTG-STORAGE-10	Testa se a aplicação executa *WebView* sem certificado de autoridade.
2.14	MSTG-STORAGE-14	Validação de *passwords* ou nomes de utilizadores armazenados em ficheiros.
Requisitos de criptografia		
#	MSTG-ID	Abordagem de automatização
3.1	MSTG-CRYPTO-1	Valida se não existem chaves SQL *Hardcoded*.
3.2	MSTG-CRYPTO-2	Testa se a aplicação usa criptografia RSA sem preenchimento de Optimal asymmetric encryption padding (OAEP) e procura os valores de instanciação da cifra RSA.
3.3	MSTG-CRYPTO-3	Verifica se os valores dos vectores de inicialização de cifra são fracos.
3.4	MSTG-CRYPTO-4	Valida o uso de algoritmos criptográficos fracos como por exemplo md4, rc2 e rc4.
3.6	MSTG-CRYPTO-6	Determina se na necessidade de usar um número aleatório, este não é gerado por uma biblioteca insegura.
Requisitos de autenticação e gestão de sessão		
#	MSTG-ID	Abordagem de automatização
4.5	MSTG-AUTH-5	Verifica se os mecanismos de validação de *password* forte estão a ser realizados no lado do dispositivo.
4.7	MSTG-AUTH-7	Estes três requisitos requerem que a aplicação use os mecanismos de autenticação por impressão digital; se a permissão não estiver declarada, não será possível usar.
4.8	MSTG-AUTH-8	
4.9	MSTG-AUTH-9	

Tabela 3.1: Abordagem de automatização efectuada pelo ransAPK.

Requisitos de comunicação em rede		
#	MSTG-ID	Abordagem de automatização
5.1	MSTG-NETWORK-1	Verifica se a configuração da comunicação entre utilizador e servidor usam algoritmos inseguros.
5.2	MSTG-NETWORK-2	Testa se a configuração da comunicação entre utilizador e servidor cumpre com as boas práticas de segurança.
5.3	MSTG-NETWORK-3	Confirma que os certificados locais em *WebView* estão correctamente configurados.
5.4	MSTG-NETWORK-4	Valida a existência de bibliotecas que sejam usadas para SSL *pinning*.
Requisitos de interacção da plataforma		
#	MSTG-ID	Abordagem de automatização
6.4	MSTG-PLATFORM-4	Valida se o uso de *Clipboard changes* está correcto ou usa funções da biblioteca inseguras.
6.5	MSTG-PLATFORM-5	Testa se existe uso de JavaScript na aplicação.
6.6	MSTG-PLATFORM-6	Confirma se os mecanismos de escrita são directos.
6.7	MSTG-PLATFORM-7	Valida se existe a implementação de JavaScript em *WebView* e é insegura.
6.8	MSTG-PLATFORM-8	Valida se a aplicação usa bibliotecas vulneráveis que permite a execução de código arbitrário.
6.9	MSTG-PLATFORM-9	Comprova se a aplicação tem capacidade de evitar ataques de *tapjacking* procurando pela função nativa *setFilterTouchesWhenObscure*.
Qualidade de código e requisitos de configuração de construção		
#	MSTG-ID	Abordagem de automatização
7.2	MSTG-CODE-2	Determina a existência de bibliotecas para ofuscação de código ou se existem IPs contidos no código sem ofuscação.
Requisitos de resiliência		
#	MSTG-ID	Abordagem de automatização
8.1	MSTG-RESILIENCE-1	A aplicação é instalada num dispositivo *rooted* para validar se esta se defende deste potencial passo num ataque.
8.2	MSTG-RESILIENCE-2	Valida o uso de bibliotecas anti *debug*, executa a aplicação num dispositivo para extrair o código em memória.
8.3	MSTG-RESILIENCE-3	Valida se a aplica usa funções nativas para impedir dump da memória e executa a aplicação num dispositivo para tentar fazer dump da memória.
8.4	MSTG-RESILIENCE-4	A aplicação é instalada num dispositivo virtual para validar se aplicação se defende.
8.5	MSTG-RESILIENCE-5	Valida se aplicação tem a capacidade de detectar um servidor Frida contido no dispositivo.
8.6	MSTG-RESILIENCE-6	Instala a aplicação num dispositivo e tenta interagir através de um servidor contido num dispositivo.
8.7	MSTG-RESILIENCE-7	Valida se a app usa bibliotecas para executar código como super utilizador (*sudo*).

Tabela 3.2: Abordagem de automatização efectuada pelo ransAPK.

3.3 Arquitectura do ransAPK

A Figura 3.1 representa a arquitectura do sistema ransAPK, a qual é composta por 9 módulos. De uma forma geral, para que o ransAPK verifique se a aplicação obedece aos requisitos de segurança é necessário obter o código da aplicação a partir dos ficheiros ".Dex". Estes ficheiros contêm os métodos da aplicação escritos em Java *Bytecode*. Com a extracção destes ficheiros é possível traduzi-los para uma linguagem de alto nível, no nosso caso a linguagem Java. Posteriormente, sobre o código, são aplicados diferentes tipos de análise estática e dinâmica para testar a presença de requisitos.

Os módulos do ransAPK são independentes uns dos outros, o que leva a que estes possam ser substituídos sem colocar em causa o funcionamento do restante sistema. Esta independência foi pensada com o intuito de dar a possibilidade aos utilizadores do sistema

de implementarem outros módulos de análise e integrá-los com o ransAPK.

O sistema é divido em quatro fases de funcionamento, onde actuam os diferentes módulos. As fases são denominadas por **Identificação do APK**, **Descompilação do APK**, **verificação de requisitos de segurança** e **Verificador de resultados**, e são descritas de seguida.

A primeira fase, **Identificação do APK**, tem a responsabilidade de identificar as capacidades de uma aplicação contra técnicas de engenharia inversa, ou seja, se o código de aplicação está protegido (*Packed*) ou não. O resultado desta verificação é comunicado à próxima fase para que esta seleccione o método a aplicar para obter os ficheiros ".Dex" da aplicação. Esta verificação é realizada pelo módulo *Identificador de APK protegida*, o qual após efectuar um *scan* ao APK verifica se o resultado contém a palavra *Packed*.

A segunda fase, **Descompilação do APK**, tem por objectivo recolher o código Java da aplicação. Para tal extrai os ficheiros ".Dex" para depois convertê-los em Smali e posteriormente em Java. Esta fase tem dois modos de operação distintos, dependendo da informação obtida da primeira fase. Caso não seja identificada nenhuma protecção no código, ou seja, caso a aplicação não esteja *Packed*, o APK será descompactado sem recorrer à sua execução num dispositivo físico. Caso a aplicação esteja *Packed*, é necessário executar o APK num dispositivo para que o seu código seja colocado em memória, e assim extrair os ficheiros ".Dex" gerados. Para concretizar esta segunda fase é necessário um dispositivo físico ou emulado.

O código da aplicação é colocado em memória quando a aplicação é executada pela primeira vez (mais detalhes na Secção 2.1.2). Se usarmos um dispositivo posterior a 2010 ou que o seu *frimware* seja posterior a esta data, asseguramos que este é AOT (Ahead-Of-Time). Garantindo estas características, após a instalação e primeira execução da aplicação, os ficheiros ".Dex" podem ser extraídos da memória. Nesta fase funcionam os módulos *Exportador de ".Dex"*, *Instalador do APK* e *Conversor de "Dex"*.

Na terceira fase, **Verificação de requisitos de segurança**, o módulo *Verificador de requisitos* aplica um conjunto de regras que irão confirmar se o código Java obedece aos requisitos de segurança presentes nas Tabelas 3.1 e 3.2. Os resultados das verificações são armazenados numa base de dados para serem processados e apresentados na quarta fase. Estes resultados têm dois valores possíveis, passou ou não passou o teste. Para cada regra, também são armazenadas outras informações relevantes sobre o requisito como, por exemplo, a evidência que justifica a decisão de passagem ou não da regra.

Na quarta fase, **Verificação de resultados**, é calculada e apresentada a maturidade da aplicação. Para calcular este valor é necessário obter da base de dados os estados de cada requisito para uma determinada aplicação. A interface do ransAPK permite ao analista alterar manualmente os valores associados aos requisitos e recalcular o valor de maturidade da aplicação. Esta capacidade pode ser útil caso o analista queira remover certos requisitos dos cálculos.

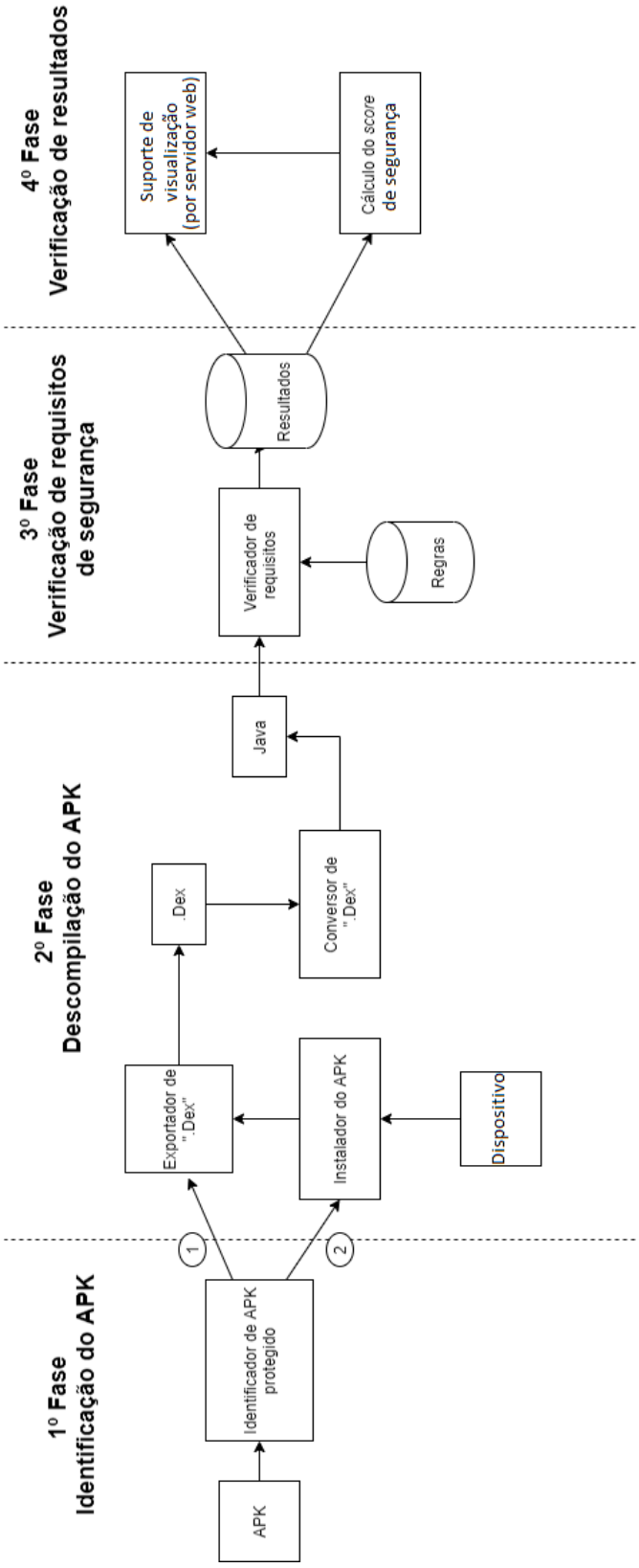


Figura 3.1: Visão geral do ransAPK.

O valor de maturidade da aplicação reflecte o estado do código relativo à implementação dos requisitos de segurança em análise. Este é expresso por um *score* que é obtido com base nos resultados obtidos (Passou ou não-Passou) sobre os requisitos e ordenados numa escala entre 0-5, onde 0 indica não implementação de requisitos e 5 um elevado estado de maturidade. Para o seu cálculo são usadas as classificações das 8 categorias de requisitos (CR_i) (ver Tabelas 3.1 e 3.2) segundo a Fórmula 1 da Figura 3.2, que expressam a percentagem de requisitos que foram verificados com sucesso (estado Passou). Na fórmula, para uma dada categoria de requisitos, P_P é o número de requisitos verificados com sucesso e P_F o número de requisitos falhados.

$$CR_i = \frac{P_P}{P_P + P_F}$$

Figura 3.2: Formula 1 - Cálculo da maturidade de um conjunto de requisitos.

Após serem obtidos os CR_i para as 8 categorias de requisitos, o valor de maturidade da aplicação é obtido pela média dos CR_i multiplicada por cinco (Formula 2 da Figura 3.3), para que o valor de maturidade esteja contido num intervalo entre zero e cinco.

$$\textit{Score} = \frac{\sum_{i=1}^{8} CR_i}{8} \times 5$$

Figura 3.3: Formula 2 - Cálculo da maturidade de segurança de uma aplicação.

3.4 Conclusão

O capítulo especificou as propriedades do sistema ransAPK e a abordagem de análise quando a aplicação apresenta mecanismos de protecção. Também descreveu a arquitectura do sistema ransAPK e o *workflow* do processamento dividido em 4 fases. A segunda fase do processamento é a mais importante: caso a tradução do Java *Bytecode* não seja bem realizada, as regras de validação que são executadas na fase seguinte, podem gerar resultados não desejados, aumentando o número de falsos positivos.

O próximo capítulo explica em maior detalhe cada uma das fases descritas, desde a fase de *Identificação do APK* até à fase de *Verificador de resultados*.

Capítulo 4

Implementação

No capítulo anterior foi apresentada a arquitectura do sistema ransAPK, assim como todas as fases de funcionamento da mesma, para que fosse possível entender as funcionalidades inerentes a cada fase do ransAPK de uma forma abstracta, mas explícita, e os módulos que nelas participam.

Neste capítulo é descrita, em detalhe, a implementação do ransAPK nas diversas fases e os seus componentes individuais. Para cada uma das fases são apresentados os módulos de *Software* utilizados, bem como os desenvolvidos, e o seu *workflow* tendo por base a Figura 4.1. O sistema foi implementado em Python 2.0 e opera no sistema operativo Linux. O sistema utiliza a ferramenta MobSF (ver Secção 2.6) para verificação de alguns requisitos de segurança. No entanto, para alguns deles as regras do MobSF foram melhoradas ou estendidas para uma melhor precisão. Para a validação dos 34 requisitos contidos na Tabela 3.1, o ransAPK integra 53 regras, onde 39 são do MobSF, 11 foram criadas de raiz para tratar de requisitos não contemplados pelo MobSF, e 3 são do MobSF, mas aperfeiçoado por nós.

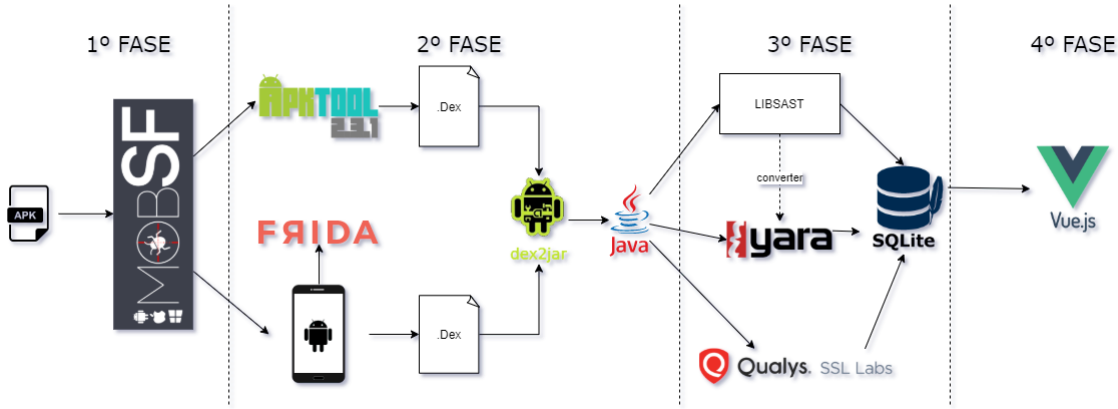


Figura 4.1: Módulos de *Software* referida na implementação do ransAPK.

4.1 1ª Fase - Identificação do APK

Como explicado no Capítulo 3, o objectivo desta fase é apurar se o código da aplicação a analisar está protegido ou não, uma vez que a validação das regras é realizada sobre o código Java. Neste sentido, é necessário obter o código sem protecção. O módulo *Identificador de APK protegido*, da Figura 3.1, é implementado através do MobSF e os seus resultados.

Em mais detalhe, quando é verificada a existência de uma nova aplicação para análise, esta é submetida ao MobSF usando a API disponibilizada pelo mesmo, que permite enviar o ficheiro para o URL http://localhost:8000/api/v1/upload. Para o MobSF iniciar a sua análise, é necessário realizar outra chamada à API no URL http://localhost:8000/api/v1/scan. Esta API é responsável pela execução das tarefas de análise estática, sendo uma delas verificar se o APK tem o código protegido. Após executada a análise, o ransAPK executa uma última chamada à API do MobSF no URL http://localhost:8000/api/v1/report_JSON para obter o resultado da verificação sob o formato de JSON.

A informação disponibilizada pelo JSON é muito importante para o ciclo de execução do ransAPK. Se este contiver nas observações a palavra *"packed"*, a aplicação tem o seu código protegido e só quando a aplicação for executada é que será possível obter o seu código sem protecção. Nesta situação, o MobSF não tem a capacidade de aplicar as suas regras para validação de requisitos.

Para contornar estas limitações, o ransAPK oferece uma solução que passa por enviar a aplicação para um dispositivo virtual/real para ser executada e extrair o seu código. Esta solução será explicada com maior detalhe na Secção 4.2.2 (Dispositivo *Android* virtual/-real e Servidor Frida).

4.2 2ª Fase - Descompilação do APK

O objectivo desta fase é extrair o código Java da aplicação em análise. O código fonte é obtido de duas formas, dependendo se o código da aplicação está protegido ou não. Caso não esteja protegido, os módulos *Exportador de ".Dex"* e *Conversor de ".Dex"* são executados, onde as duas implementações são explicadas nas Secções 4.2.1 e 4.2.3. Caso contrário, os módulos *Instalador do APK* e *Conversor de ".Dex"* são executados. As Secções 4.2.2 e 4.2.3 descrevem a implementação desta segunda solução.

4.2.1 MobSF - APKtool

Para executar as regras do MobSF são necessários dois processos, nomeadamente a extracção e a conversão. O processo de extracção serve para obter os ficheiros ".Dex" contidos no APK. Para tal, o MobSF utiliza a APKtool [1], uma ferramenta para aplicações

binárias de *Android* que permite descompilar APK's para a forma quase original, e reconstruí-las após fazer algumas modificações.

Os ficheiros extraídos pelo APKtool são os ficheiros ".Dex" e XML, os quais contêm o código necessário para a análise estática.

4.2.2 Dispositivo *Android* virtual/real e Frida

Conforme foi explicado na Secção 4.1, quando a aplicação está *Packed* (com o código protegido) é necessário desempacotar a aplicação, ou seja, extrair o seu código quando este é colocado em memória durante a sua execução. Para tal, é necessário instalar a aplicação num dispositivo *Android* físico ou virtual e usar a ferramenta Frida [11].

O Frida é um *Greasemonkey* para aplicações ou, em termos mais técnicos, é um kit de ferramentas para a introdução de código dinâmico, permitindo que se injecte *snippets* de JavaScript ou a sua própria biblioteca em aplicações nativas no Windows, macOS, GNU / Linux, iOS, *Android* e QNX.

Para utilizar o Frida é necessário instalar um servidor dentro de um dispositivo físico ou emulado. Depois de o servidor estar em execução, usamos o adb [13] do *Android* para interagir com o servidor. Existem várias bibliotecas com código-aberto para a extracção de código carregado em memória. O ransAPK utiliza a FRIDA-DEXDump [16], ou seja, envia o código JavaScript para ser interpretado pelo Frida e o resultado desta interpretação são os ficheiros ".Dex". Este processo é automatizado por um *script* do ransAPK.

O *script* do ransAPK pode ser visto na Figura 4.2. É de realçar que as linhas de código que realizam o trabalho descrito anteriormente são as linhas entre 16-21. A linha 17 corresponde à instalação da aplicação no dispositivo, a linha 18 juntamente com a linha 16 executam o servidor Frida. Este é executado com privilégios de *root* para garantir que a extracção de código vai ser realizada num dispositivo *rooted*. Nas linhas 19, 20 e 21 é executada a aplicação em análise e por fim é extraído todo o seu código.

4.2.3 MobSF - Dex2Jar

Dex2Jar [5] é uma ferramenta para obter o código Java, através da tradução dos ficheiros ".Dex". É uma biblioteca de código-aberto e recebe actualizações constantes de melhoria, tendo sido este o principal motivo para ter sido integrada no ransAPK. Nos dois fluxos de extracção de código, que podem ser vistos na Figura 4.1, a ferramenta opera de maneira semelhante. São executados exactamente os mesmos comandos, distinguindo-se apenas a proveniência dos ficheiros ".Dex" para a sua execução: se a aplicação não estiver protegida é da responsabilidade do APKtool entregar os ficheiros ".Dex", mas se esta estiver protegida é o Frida que deve disponibilizar esses ficheiros (ver secção anterior).

```
import os
import time
from os import system

PRODUCTION = False
UNPACKER = "/libraries/Frida-DEXDump-master"

class decompile\_app:

def __init__(self, APK, package,activity):
    print "Packed App detected, INFO:"
    print "APK:"+apk
    print "Package:"+package
    print "Activity:"+activity
    dir_path = os.path.dirname(os.path.realpath(__file__))
    cmd = "su -c"
    system("adb install {}".format(apk))
    system("adb shell {} data/local/tmp/Frida-server-12.10.4-android-
    arm \& ".format(cmd))
    system("Frida --no-pause -U -f {} \&".format(package))
    time.sleep(10)
    system("python {}/main.py".format(dir_path+UNPACKER))
```

Figura 4.2: *Script* que extrai os ficheiros ".Dex" pelo Frida.

4.3 3ª Fase - Verificação de requisitos de segurança

O objectivo da 3ª fase é verificar se os requisitos de segurança estão satisfeitos pela aplicação em análise. O módulo *Verificador de requisitos* (ver Figura 3.1) é responsável por esta tarefa e é implementado através das regras contidas no MobSF (Secção 4.3.1), das regras criadas pelo ransAPK (Secção 4.3.2) e por um *Software* para verificação de comunicação segura em rede (Secção 4.3.3).

4.3.1 Regras - Libsast

A Libsast é uma biblioteca criada pelos autores do MobSF, que tem o objectivo de detectar padrões de código. A análise é feita da seguinte forma: o conteúdo do ficheiro a examinar é colocado dentro de um *array*, e verifica-se se as regras correspondem a algum do conteúdo. A maioria dos requisitos que são testados pelo ransAPK (23 de 34) são verificados por esta biblioteca. A Tabela 4.1 mostra em detalhe os requisitos verificados (ver 3ª coluna).

4.3.2 Regras - Yara

Um conjunto extra de regras foi criado para o ransAPK para tratar dos requisitos que o MobSF não verifica ou para complementar os testes restantes, de forma a melhorar a sua

precisão. As regras foram escritas com recurso ao *Software* Yara [40]. O Yara permite identificar Malware (ou outros arquivos) através da criação de regras que procuram determinadas características. Neste sentido, as regras descrevem padrões que identificam fluxos de código malicioso ou famílias inteiras de Malware. Com base neste conceito, foram criadas 6 regras Yara para verificar 10 requisitos. A Tabela 4.1, nas colunas 4 e 5, mostra quais as regras que foram melhoradas e quais foram criadas para o ransAPK.

Este módulo do ransAPK é totalmente extensível. Se o analista quiser colocar mais regras Yara basta criar um ficheiro de extensão ".yara" e colocar na pasta "backend/yara_rules". O nome do ficheiro deve ser igual ao identificador referente ao requisito a verificar.

Conforme dito anteriormente, o MobSF não tem a capacidade de executar as suas regras quando uma aplicação está protegida, mas o ransAPK consegue. Para tal, as regras usadas pelo MobSF são importadas para o ransAPK sob forma de regras Yara e executadas posteriormente.

Os requisitos validados pelas regras Yara criadas para o ransAPK são apresentados de seguida, por categoria de requisitos.

Requisitos de arquitectura, desenho e modelação de ameaças

O código apresentado na Figura 4.3 fornece um exemplo da utilização da regra MSTG-ARCH-9 para validar se a biblioteca *AppUpdateManager* é usada e se chama a função necessária para a validação da versão actual do APK. Para testar o requisito é necessário que os três nomes listados nas linhas 4-6 sejam incluídos no código da aplicação.

```
1 rule MSTGARCH9
2 {
3 strings:
4     $class = "AppUpdateManager"
5     $service = "AppUpdateManagerFactory"
6     $instance = "AppUpdateInfo"
7 condition:
8     $class and $service and $instance
9 }
```

Figura 4.3: Regra MSTG-ARCH-9.

Requisitos de armazenamento de dados e requisitos de privacidade

Quando os utilizadores escrevem texto nos campos de entrada, a aplicação pode sugerir formas de completar as frases. Este recurso pode ser muito útil, por exemplo, para aplicações de envio de mensagens. No entanto, a *cache* do teclado pode divulgar informações confidenciais quando o utilizador selecciona um campo de entrada que terá esse tipo de informação.

Na definição da organização de uma actividade, as *TextViews* que possuem atributos como XML *Android inputType*, devem definir o atributo *textNoSuggestions*. Nesta situação a *cache* do teclado não será mostrada quando o campo de entrada for seleccionado e o utilizador terá de digitar tudo manualmente. Para detectar se uma aplicação cumpre este requisito, foi criada a seguinte regra Yara (ver Figura 4.4):

```
rule MSTGSTORAGE5
{
strings:
    $xml_atribute_1 = "KeyBoardCache"
    $xml_atribute_2 = "textNoSuggestions"
condition:
$xml_atribute_1 and $xml_atribute_2
}
```

Figura 4.4: Regra MSTG-STORAGE-5.

Requisitos de autenticação e gestão de sessão

Nesta categoria, o ransAPK valida os requisitos MSTG-AUTH-5, MSTG-AUTH-7, MSTG-AUTH-8 e MSTG-AUTH-9. Isto é feito através de duas regras Yara:

- MSTG-AUTH-5: o objectivo é verificar se existem mecanismos que obriguem o utilizador a ter uma *password* forte. As expressões regulares são frequentemente usadas para validações deste tipo e também são as recomendadas pela OWASP. Se o programador verificar localmente a complexidade da *password* é possível detectar com a seguinte regra (ver Figura 4.5);
- MSTG-AUTH-7, MSTG-AUTH-8 e MSTG-AUTH-9: requerem o uso de autenticação biométrica. A regra definida na Figura 4.6 detecta o incumprimento através da análise do ficheiro *manifest*. Caso esta forma de autenticação não esteja definida no ficheiro, estes três requisitos falham por omissão.

```
rule MSTGAUTH5
{
strings:
    $string = "The password must"
    $regular_1 = "(.)\\1{2,}"
    $regular_2 = "a-z"
    $regular_3 = "A-Z"
    $regular_4 = "0-9"
    $regular_5 = "^A-Za-z0-9"
condition:
   $regular_1 or
   $regular_2 and
   $regular_3 and
   $regular_4 and
   $regular_5 and
   $string
}
```

Figura 4.5: Regra MSTG-AUTH-5.

```
rule MSTGAUTH8
{
strings:
   $permission = "android.permission.USE_FINGERPRINT"
condition:
   $permission
}
```

Figura 4.6: Regra MSTG-AUTH-8.

Requisitos de interacção da plataforma

Várias configurações podem ser seleccionadas no *WebView*, como activar ou desactivar o JavaScript. O JavaScript está desligado por omissão para as *WebViews* e para que possa ser usado, necessita de ser explicitamente configurado. Para detectar se uma aplicação cumpre o requisito MSTG-PLATFORM-5 foi criada a regra Yara apresentada na Figura 4.7. Caso seja detectado que o JavaScript está activo a aplicação falha o requisito.

```
rule MSTGPLATFORM5
{
strings:
    $Java_script_true = "setJavaScriptEnabled(true)"
    $Java_script_1 = "setJavaScriptEnabled(1)"
condition:
    $Java_script_1 or $Java_script_true
}
```

Figura 4.7: Regra MSTG-PLATFORM-5.

Quando se usa *addJavascriptInterface*, os métodos Java só são acedidos pelo JavaScript quando a anotação *@JavascriptInterface* é adicionada. Antes da API de nível 17, todos os métodos Java eram acedidos por defeito. Uma aplicação com uma versão do *Android* anterior à *Android 4.2* é vulnerável à falha em *addJavascriptInterface* e não deve ser utilizada. Na Figura 4.8 temos a definição desta regra.

```
rule MSTGPLATFORM7
{
strings:
    $Java_script_true = "setJavaScriptEnabled(true)"
    $Java_script_1 = "setJavaScriptEnabled(1)"
    $Java_class = "addJavascriptInterface"
condition:
    ($Java_script_1 and $Java_class) or
    ($Java_script_true and $Java_class)
}
```

Figura 4.8: Regra MSTG-PLATFORM-7.

Requisitos de resiliência

Para validar um requisito, o MobSF apenas usa análise estática. Esta validação pode produzir resultados errados por várias razões, por exemplo, o código que foi verificado por uma regra pode não ser executado. Com a análise dinâmica, as validações não correm este risco porque as regras são validadas sobre o código que está em execução. O ransAPK consegue melhorar os pontos MSTG-RESILIENCE-1 e MSTG-RESILIENCE-2, e acrescenta outros dois. Esta validação é feita sobre o resultado do *script*, ou seja, se o *script* conseguir correr na sua totalidade a aplicação falha os quatro requisitos mencionados. Caso aconteça algum erro durante a execução do *script*, então não será assinalado nenhum resultado.

Se observarmos os requisitos MSTG-RESILIENCE-1, MSTG-RESILIENCE-2, MSTG-RESILIENCE-5 e MSTG-RESILIENCE-6, estes validam se a aplicação detecta que o dispositivo está *rooted*, se a aplicação detecta tentativas de forçar o *debug*, se a aplicação responde ao facto de estar a correr num dispositivo virtual e por fim se esta ofusca o código colocado em memória. Validar estes requisitos um a um pode resultar em falsos positivos, isto porque existem aplicações que não impedem a instalação em dispositivos *rooted* ou emulados, apenas não executam, ou mostram uma mensagem de erro. Mas, por outro lado, é possível concluir que se o ransAPK extrair o código da aplicação e se este foi extraído de um dispositivo virtual e *rooted*, então estes 4 requisitos falham. Acrescento que interacções do Frida com uma aplicação permite realizar o *debug* da mesma.

#	MSTG-ID	MobSF	MobSF melhorado	ransAPK	Nº de regras
Requisitos de arquitectura, desenho e modelação de ameaças					
1.9	MSTG-ARCH-9			X	1
Armazenamento de dados e requisitos de privacidade					
2.2	MSTG-STORAGE-2	X			4
2.5	MSTG-STORAGE-5			X	1
2.7	MSTG-STORAGE-7	X			1
2.9	MSTG-STORAGE-9	X			2
2.10	MSTG-STORAGE-10	X			1
2.14	MSTG-STORAGE-14	X			1
Requisitos de criptografia					
3.1	MSTG-CRYPTO-1	X			3
3.2	MSTG-CRYPTO-2	X			1
3.3	MSTG-CRYPTO-3	X			2
3.4	MSTG-CRYPTO-4	X			4
3.6	MSTG-CRYPTO-6	X			1
Requisitos de autenticação e gestão de sessão					
4.5	MSTG-AUTH-5			X	1
4.7	MSTG-AUTH-7			X	1
4.8	MSTG-AUTH-8			X	1
4.9	MSTG-AUTH-9			X	1
Requisitos de comunicação em rede					
5.1	MSTG-NETWORK-1			X	1
5.2	MSTG-NETWORK-2			X	1
5.3	MSTG-NETWORK-3	X			1
5.4	MSTG-NETWORK-4	X			1
Requisitos de interacção da plataforma					
6.4	MSTG-PLATFORM-4	X			1
6.5	MSTG-PLATFORM-5			X	1
6.6	MSTG-PLATFORM-6	X			1
6.7	MSTG-PLATFORM-7	X	X	X	2
6.8	MSTG-PLATFORM-8	X			1
6.9	MSTG-PLATFORM-9	X			1
Qualidade de código e requisitos de configuração de construção					
7.2	MSTG-CODE-2	X			2
Requisitos de resiliência					
8.1	MSTG-RESILIENCE-1	X	X	X	3
8.2	MSTG-RESILIENCE-2	X	X	X	3
8.3	MSTG-RESILIENCE-3	X			3
8.4	MSTG-RESILIENCE-4	X			2
8.5	MSTG-RESILIENCE-5			X	1
8.6	MSTG-RESILIENCE-6			X	1
8.7	MSTG-RESILIENCE-7	X			1

Tabela 4.1: Regras implementadas no ransAPK.

4.3.3 SSLabs

Uma das principais funções de uma aplicação é enviar e receber dados, ocorrendo isto muitas vezes em redes não confiáveis, como a Internet. Se os dados não estiverem devidamente protegidos em trânsito, então um invasor com acesso a qualquer parte da rede (por exemplo, um ponto de acesso Wi-Fi) pode interceptar, ler ou modificá-los.

A grande maioria das aplicações utiliza o protocolo HTTP para comunicação com os servidores. O protocolo TLS permite a autenticação do serviço remoto e garante a confidencialidade e integridade dos dados na rede. Garantir a configuração adequada de TLS no lado do servidor também é importante. As versões v1.2 e v1.3 do TLS são consideradas seguras, mas muitos serviços ainda permitem a utilização das versões v1.0 e v1.1 para compatibilidade com clientes mais antigos. Estas versões são desaconselhadas devido a limitações de segurança. Se uma aplicação *Android* se conectar a um servidor específico, a sua pilha de protocolos de rede pode ser ajustada para garantir o nível de segurança mais elevado possível para a configuração do servidor. A falta de suporte no sistema operativo subjacente pode forçar a aplicação *Android* a usar uma configuração mais fraca.

Para testar os requisitos MSTG-NETWORK-1 e MSTG-NETWORK-2 da categoria de comunicação em rede, foi usada uma API externa disponibilizada pelo SSLabs. Esta API tem a capacidade de validar quais as configurações de TLS de um servidor remoto. Após a validação, é atribuída uma classificação entre A e F, sendo A a melhor classificação possível e F a pior. Se for detectado que a aplicação comunica com um servidor de classificação inferior a A, a aplicação irá falhar estes requisitos.

4.3.4 Base de dados - SQLite

De forma a guardar os resultados das análises efectuadas pelo ransAPK foi implementada uma base de dados SQLite. Este sistema de base de dados foi escolhido pela sua simplicidade e por fazer parte da distribuição base de Python.

O esquema da base de dados é composto por duas tabelas. A tabela *Results* contém 7 campos e a tabela *Application* contém 11 campos. A Figura 4.9 ilustra ambas as tabelas e como estão ligadas. É da responsabilidade da tabela *Application* guardar os dados que correspondem à informação da aplicação analisada, sendo posteriormente consumidos pela interface do ransAPK (ver Secção 4.4.1). Para a chave primária da tabela *Application* foi escolhido o *hash* do APK, suportando a criação de análises com novas versões da mesma aplicação sem nenhum impedimento. A tabela *Results* guarda os requisitos analisados e não testados, fazendo-se a distinção através do campo *status*, do tipo booleano. O valor deste booleano é transparente para o utilizador, pois serve para construir as tabelas que são apresentadas pela interface do ransAPK.

De forma a não criar incoerências na base de dados, sempre que se voltar a analisar

uma aplicação com a mesma *hash*, os valores de entrada contidos nas tabelas *Application* e *Results* sobre esta aplicação são removidos.

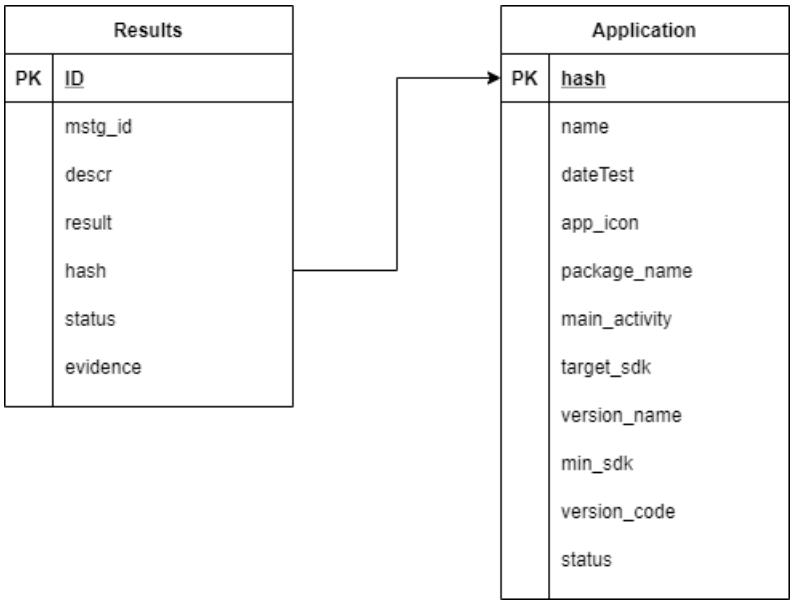


Figura 4.9: Esquema de base de dados do ransAPK.

4.4 4ª Fase - Verificação de resultados

Esta fase tem como objectivo apresentar os resultados da análise de requisitos e calcular o valor da maturidade da aplicação. Neste sentido, foi implementada uma interface web (Secção 4.4.1) para visualização dos resultados e interacção com o utilizador, e o módulo *Cálculo do score de segurança* (Secção 4.4.2).

4.4.1 Interface com o utilizador

O ransAPK oferece uma interface desenvolvida em VueJs [37], onde são reorganizadas todas as análises efectuadas, permitindo que a interacção entre o utilizador e os objectos de teste seja facilitada.

A página principal (ver Figura 4.10) do ransAPK contém uma zona onde pode ser efectuado o carregamento de uma aplicação *Android*, bem como visualizar aplicações previamente analisadas. A partir desta página, o analista pode apagar ou adicionar novas aplicações.

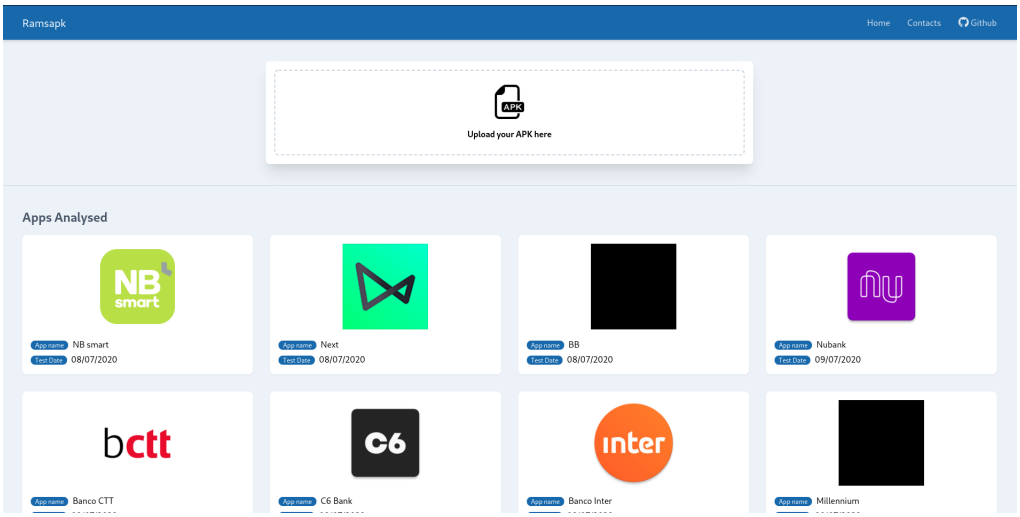


Figura 4.10: Página inicial do ransAPK.

Ao seleccionar uma aplicação, o analista é direccionado para a página do respectivo APK (ver Figura 4.11). Esta página apresenta várias informações, nomeadamente:

- Nome da aplicação;
- *Hash* do APK analisado;
- Data da realização do teste;
- Nome da primeira actividade (*Main activity*) a ser executada na aplicação;
- Nome do pacote da aplicação;
- SDK alvo e alvo mínimo;
- Versão de código de acordo com o especificado no ficheiro *manifest*;
- Estado dos requisitos (*Pass, Fail e Not Applicable*);
- *Score* de maturidade da aplicação.

Existem informações estáticas e dinâmicas nesta página. As informações dinâmicas são o estado dos requisitos e o *score*, onde o primeiro pode ser alterado pelo analista, de acordo com a relevância do requisito. O segundo, é recalculado com base nos novos estados dos requisitos.

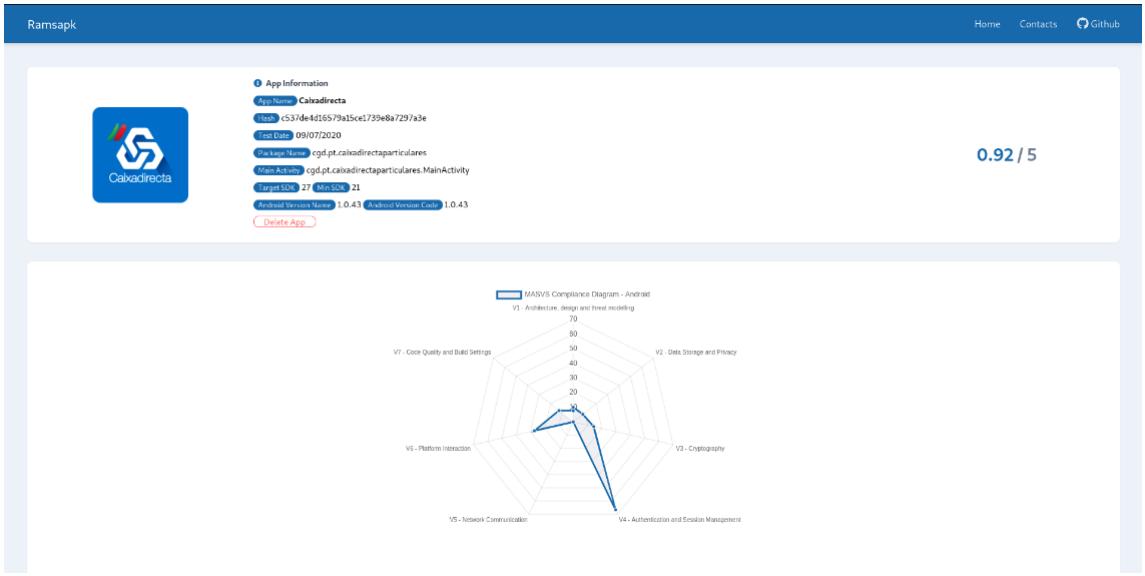


Figura 4.11: Página de análise.

Se o analista quiser validar um requisito, isto é, alterar o seu estado pode fazê-lo a partir da interface, que permite a selecção entre os valores *Pass, Fail e Not Applicable*, para cada requisito.

4.4.2 *Score* de maturidade da aplicação

A fórmula para o cálculo do *score* de maturidade de uma aplicação analisada foi discutida na Secção 3.3, e é efectuado pela interface do ransAPK. Quando existe a necessidade de calcular este valor, a interface realiza um pedido à base de dados e esta devolve todos os resultados armazenados para a aplicação pedida. Para a obtenção dos resultados de uma determinada aplicação, o ransAPK dispõe de uma API oferecida por um servidor. Nesta situação é necessário chamar o serviço "/api/v1/results/" onde os parâmetros a enviar incluem o *hash* da aplicação para a qual se pretende obter o resultado. A Figura 4.12 ilustra o pedido que devolve os resultados da tabela *Results*. A Figura 4.13, apresenta o código necessário para se obter o resultado da maturidade de uma aplicação.

```
1 @app.route('/api/v1/results/<hash>', methods = ['GET'])
2 def get_result(hash):
3     dict_ = do_select("select * from Results where hash = '" + hash + "
    '")
4     return JSONify(dict_)
```

Figura 4.12: Função get_result.

```
async getResults () {
  this.graphValues = null
  const data = await this.fetch_results_application(this.id)
  const temp = this.results = _.chain(data)
    .filter(item => _.chain(item).get('mstg_id').split('-').size().value() === 3)
    .orderBy('mstg_id')
    .value()

  this.results = _.chain(this.description_headers)
    .map((i) => {
      _.set(i, 'childs', _.filter(temp, j => _.startsWith(j.mstg_id, i.mstg_id)))
      return i
    })
    .map((i) => {
      _.set(i, 'pass', this.getTotalPass(i))
      _.set(i, 'fail', this.getTotalFail(i))
      _.set(i, 'na', this.getTotalNa(i))
      _.set(i, 'pct', this.get_decimal((this.getTotalPass(i) / (this.getTotalPass(i) + this.getTotalFail(i))) * 100) || 0)
      return i
    })
    .value()

  this.results = _.chain(this.results)
    .map((item) => {
      item.childs = _.map(item.childs, (j) => {
        _.set(j, 'mstg_id_descr', _.chain(this.descriptions).find(k => k.mstg_id === j.mstg_id).get('descr').value())
        return j
      })
      return item
    })
    .value()

  this.graphValues = _.map(this.results, 'pct') || 0
  this.globalScore = this.get_decimal(_.sum(this.graphValues) / 100 / 8 * 5) || 0
}
```

Figura 4.13: Método get_result.

4.5 Conclusão

O ransAPK apresenta na sua implementação uma funcionalidade completamente distinta do MobSF, nomeadamente a capacidade de desempacotar as aplicações com o seu código protegido, e com isto fornecer uma análise mais fidedigna. O sistema também acrescentou a capacidade de verificar mais 11 requisitos não tratados pelo MobSF. Entre estes requisitos, 7 são concretizados por regras Yara, 2 são validados com recurso a uma API externa e 2 são testados com análise dinâmica. Existem ainda 3 requisitos que foram melhorados,

2 pela capacidade da análise dinâmica e 1 pela análise estática.

Na implementação do ransAPK, existem dois conjuntos de requisitos para os quais não foram adicionadas regras (para além das contidas no MobSF), sendo estes relacionados com *Qualidade de código e requisitos de configuração de construção* e *Criptografia*. Os requisitos de *Qualidade de código e requisitos de configuração de construção* devem ser validados na origem do código, e como tal não foram adicionados. Relativamente aos requisitos de *Criptografia*, dada a qualidade de cobertura do MobSF, também não foram contemplados pelos ransAPK.

Resumidamente, o ransAPK foi criado para ser o mais abrangente possível na implementação dos requisitos OWASP, facilitando a consulta de resultados e o trabalho dos analistas.

Capítulo 5

Resultados

Neste capítulo serão apresentados os resultados do ransAPK ao analisar várias aplicações, pertencentes a três categorias: rastreio COVID-19, bancárias e *Packed*. A variedade de aplicações pretende demonstrar a versatilidade do sistema e como este consegue processar vários tipos de aplicação, seja de exigência L1, L2 ou RE (Secção 2.3). Ainda neste capítulo, iremos comparar o ransAPK e MobSF, ao estudarem uma aplicação *Packed*, isto é, com o seu código protegido.

5.1 Selecção de aplicações

As aplicações foram escolhidas de diferentes contextos e criticidade para mostrar a versatilidade do ransAPK, sendo seleccionadas: aplicações usadas no sector bancário, de rastreio de COVID-19 e com o código protegido *Packed*.

As aplicações bancárias são consideradas críticas porque normalmente basta explorar uma vulnerabilidade para haver um impacto significativo no utilizador, por exemplo, a divulgação de dados pessoais. Se observamos as funcionalidades de uma aplicação que represente um banco, podemos facilmente identificar um grande uso de dados privados e confidenciais, por isso é esperado que estes tipos de aplicações cumpram os requisitos L2 + RE.

As aplicações COVID-19 foram criadas com o intuito de rastrear a propagação do vírus, e não necessitam de qualquer tipo de dado privado e confidencial. Face ao estado actual da pandemia, o uso deste tipo de aplicações tem sido recomendado por vários governos, tornando-as relevantes do ponto de vista de segurança. Estas aplicações devem cumprir requisitos L1 + RE, por isso foram seleccionadas para análise do ransAPK.

Por fim, foi seleccionada uma aplicação de código seguro *Packed* com o objectivo de comparar as capacidades do sistema ransAPK, face ao MobSF. No total foram seleccionadas 17 aplicações, nomeadamente 12 bancárias, 4 de rastreio COVID-19 e uma *Packed*. A Tabela 5.1 lista estas aplicações.

Aplicação	Descrição	Nível de segurança expectado	código protegido *Packed*
NB Smart	Aplicação de Mobile Banking do NOVO BANCO e NOVO BANCO dos Açores, para clientes particulares e Empresas	L2 + RE	Não
CA Mobile Empresas	Aplicação de Mobile Banking do Crédito Agrícola exclusivo para empresas	L2 + RE	Não
DABOX	A DABOX é uma app da Caixa que disponibiliza os serviços de informação sobre contas à ordem e de iniciação de pagamentos.	L2 + RE	Não
Barclays	A aplicação Barclays Mobile Banking permite que fazer pagamentos, mover dinheiro entre contas e visualizar saldo de transacções	L2 + RE	Não
CA Mobile	Aplicação de Mobile Banking do Crédito Agrícola exclusivo para clientes	L2 + RE	Não
Millennium	Aplicação de Mobile Banking do Millennium BCP para clientes particulares e Empresas	L2 + RE	Não
Santander	Aplicação de Mobile Banking do Santander para clientes particulares	L2 + RE	Não
M24 Empresas	Aplicação de Mobile Banking do Montepio para empresas	L2 + RE	Não
Moey!	Aplicação do banco Crédito Agrícola para jovens	L2 + RE	Não
Garanti BBVA	Aplicação de Mobile Banking Turka para clientes	L2 + RE	Não
ActivoBank	Aplicação de Mobile Banking para clientes	L2 + RE	Não
Banco CTT	Aplicação de Mobile Banking para clientes	L2 + RE	Não
NHS COVID-19	Aplicação oficial decretada governo do Reino Unido para rastreio de COVID-19	L1 + RE	Não
Stayaway	Aplicação oficial decretada governo Português para rastreio de COVID-19	L1 + RE	Não
AntiCOVID	Aplicação oficial decretada governo Francês para rastreio de COVID-19	L1 + RE	Não
SocMonitoring	Aplicação oficial decretada governo Russo para rastreio de COVID-19	L1 + RE	Não
KrazyRupee	Aplicação Indiana de empréstimo pessoal instantâneo para utilizadores de nacionalidade Indiana	L2 + RE	Sim

Tabela 5.1: Aplicações analisadas pelo ransAPK.

5.2 Configuração e desempenho da solução

A configuração dos testes é idêntica ao descrito no Capítulo 4, onde é usada uma máquina com sistema operativo Linux, 4 GB RAM e 10 *cores* de processador. Foram analisadas mais de 50 aplicações ao longo do desenvolvimento do sistema ransAPK. Foi observado que, em média, o ransAPK realiza as suas funções num intervalo de 43 segundos, sendo o caso mais rápido realizado em 36 segundos, e o mais lento em 116 segundos. A fórmula usada para obter o tempo de execução, é a diferença de tempo entre a detecção no ransAPK de uma nova aplicação para análise e o registo dos resultados provenientes na base de dados. Ainda relacionado com o desempenho deste sistema, foi verificado com sucesso pelo menos um requisito em cada aplicação.

5.2.1 Aplicações bancárias

As Tabelas 5.2, 5.3 e 5.4 apresentam os resultados das 12 aplicações analisadas pelo ransAPK, sendo esperado que estas cumpram os requisitos L2 + RE. Neste tipo de aplicações todos os módulos são críticos, e um requisito que não seja cumprido pode resultar numa vulnerabilidade que exponha ou danifique informação do utilizador. As vulnerabilidades mais críticas derivam de falhas nos *Requisitos de resiliência*, *de comunicação em rede e de autenticação*. Os *Requisitos de resiliência*, se não forem cumpridos, possibilitam que um atacante infecte um dispositivo podendo executar instruções directamente na zona de memória onde estas aplicações estão a ser executadas. Esta actividade maliciosa pode resultar em acções como, por exemplo, transferências de saldos.

O conjunto de *Requisitos de comunicação em rede* garantem que o utilizador comunica em segurança entre o seu dispositivo e o servidor, impedindo que ataques como homem-no-meio [30] sejam executados com sucesso. Só assim se consegue que a informação do utilizador não seja observada por um atacante.

Os *Requisitos de autenticação* garantem que o utilizador não é personificado, e que só este tem acesso à sua informação. Se estes requisitos não estiverem bem implementados, um atacante pode ter acesso a toda informação privada e confidencial do utilizador.

Se observarmos as Tabelas 5.2, 5.3 e 5.4, é visível que a maioria das aplicações analisadas contêm uma ou mais falhas nos requisitos anteriormente descritos, sofrendo de vulnerabilidades críticas. Nas tabelas são representadas as situações de falha e de cumprimento dos requisitos, utilizando as letras “F” e “P”, respectivamente. Quando o ransAPK não é capaz de verificar determinado requisito, nada é assinalado na tabela. Tal acontece quando as regras deferidas no ransAPK não conseguem ser verificas no código da aplicação. A aplicação Moey! do Crédito Agrícola é a que apresenta menos falhas de implementação de requisitos de segurança. Em sentido inverso a aplicação do Millennium é a que apresenta o maior número de problemas. Se observarmos os resultados com foco nos *Requisitos de armazenamento de dados e de privacidade*, existe sempre pelo

menos uma falha. Isto deve-se ao facto das aplicações terem a necessidade de guardar o estado das variáveis em ficheiros temporários, mas a criação deste ficheiro nem sempre é feita de forma segura. Já na secção de *Criptografia* também existem algumas falhas detectadas. Um pouco em complemento do requisito anterior, *Armazenamento de dados e requisitos de privacidade*, o programador ao cifrar informação do estado das variáveis acaba por usar algoritmos que são vulneráveis e o atacante consegue facilmente reverter.

Os *Requisitos de interacção de plataforma* aparecem como falha de algumas aplicações bancárias. Por norma estas aplicações permitem ao utilizador visualizar um PDF onde consta a sua informação privada e algumas acções que este possa ter feito. O problema é que a visualização do forma PDF em *WebView* requer JavaScript, o que expõe o utilizador a vulnerabilidades.

#	MSTG-ID	NB smart	CA Mobile Empresas	DABOX	Barclays
Requisitos de arquitectura, desenho e modelação de ameaças					
1.9	MSTG-ARCH-9				
Armazenamento de dados e requisitos de privacidade					
2.2	MSTG-STORAGE-2	F	F	F	F
2.3	MSTG-STORAGE-3	F	F	F	F
2.5	MSTG-STORAGE-5				
2.7	MSTG-STORAGE-7				
2.9	MSTG-STORAGE-9				
2.10	MSTG-STORAGE-10	F		F	
2.14	MSTG-STORAGE-14	F	F	F	
Requisitos de criptografia					
3.1	MSTG-CRYPTO-1		F	F	
3.2	MSTG-CRYPTO-2	F			
3.3	MSTG-CRYPTO-3				
3.4	MSTG-CRYPTO-4	F	F	F	
3.6	MSTG-CRYPTO-6	F	F	F	F
Requisitos de autenticação e gestão de sessão					
4.5	MSTG-AUTH-5	F			
4.7	MSTG-AUTH-7	F			
4.8	MSTG-AUTH-8	F			
4.9	MSTG-AUTH-9	F			
Requisitos de comunicação em rede					
5.1	MSTG-NETWORK-1	F		F	F
5.2	MSTG-NETWORK-2	F		F	F
5.2	MSTG-NETWORK-3	F			F
5.2	MSTG-NETWORK-4	F	P	F	
Requisitos de interacção da plataforma					
6.4	MSTG-PLATFORM-4	F			
6.5	MSTG-PLATFORM-5	F		F	F
6.6	MSTG-PLATFORM-6				
6.7	MSTG-PLATFORM-7	F		F	
6.8	MSTG-PLATFORM-8				
6.9	MSTG-PLATFORM-9				
Qualidade de código e requisitos de configuração de construção					
7.2	MSTG-CODE-2			F	
Requisitos de resiliência					
8.1	MSTG-RESILIENCE-1	P	P	P	
8.2	MSTG-RESILIENCE-2				
8.3	MSTG-RESILIENCE-3				
8.4	MSTG-RESILIENCE-4				
8.5	MSTG-RESILIENCE-5				
8.6	MSTG-RESILIENCE-6				
8.7	MSTG-RESILIENCE-7				

Tabela 5.2: Resultado das aplicações bancárias NB smart, CA Mobile Empresas, DABOX e Barclays.

#	MSTG-ID	CA Mobile	Millennium	Santander	M24 Empresas
Requisitos de arquitectura, desenho e modelação de ameaças					
1.9	MSTG-ARCH-9				
Armazenamento de dados e requisitos de privacidade					
2.2	MSTG-STORAGE-2	F	F	F	F
2.3	MSTG-STORAGE-3	F	F	F	F
2.5	MSTG-STORAGE-5				
2.7	MSTG-STORAGE-7				
2.9	MSTG-STORAGE-9				
2.10	MSTG-STORAGE-10		F	F	F
2.14	MSTG-STORAGE-14	F	F	F	F
Requisitos de criptografia					
3.1	MSTG-CRYPTO-1	F			
3.2	MSTG-CRYPTO-2		F	F	F
3.3	MSTG-CRYPTO-3				
3.4	MSTG-CRYPTO-4	F	F	F	F
3.6	MSTG-CRYPTO-6	F	F	F	F
Requisitos de autenticação e gestão de sessão					
4.5	MSTG-AUTH-5	F			
4.7	MSTG-AUTH-7	F			
4.8	MSTG-AUTH-8	F			
4.9	MSTG-AUTH-9	F			
Requisitos de comunicação em rede					
5.1	MSTG-NETWORK-1	F	F	F	F
5.2	MSTG-NETWORK-2	F	F	F	F
5.3	MSTG-NETWORK-3		F	F	
5.4	MSTG-NETWORK-4		F	F	
Requisitos de interacção da plataforma					
6.4	MSTG-PLATFORM-4	F			
6.5	MSTG-PLATFORM-5		F	F	F
6.6	MSTG-PLATFORM-6				
6.7	MSTG-PLATFORM-7		F	F	F
6.8	MSTG-PLATFORM-8				
6.9	MSTG-PLATFORM-9				F
Qualidade de código e requisitos de configuração de construção					
7.2	MSTG-CODE-2	F	F	F	
Requisitos de resiliência					
8.1	MSTG-RESILIENCE-1	F	F	F	F
8.2	MSTG-RESILIENCE-2	F	F	F	F
8.3	MSTG-RESILIENCE-3				
8.4	MSTG-RESILIENCE-4				
8.5	MSTG-RESILIENCE-5	F	F	F	F
8.6	MSTG-RESILIENCE-6	F	F	F	F
8.7	MSTG-RESILIENCE-7				

Tabela 5.3: Resultado das aplicações bancárias CA Mobile, Millennium, Santander e M24 Empresas.

#	MSTG-ID	moey!	Garanti BBVA	ActivoBank	Banco CTT
Requisitos de arquitectura, desenho e modelação de ameaças					
1.9	MSTG-ARCH-9				
Armazenamento de dados e requisitos de privacidade					
2.2	MSTG-STORAGE-2		F	F	F
2.3	MSTG-STORAGE-3	F	F	F	F
2.5	MSTG-STORAGE-5				
2.7	MSTG-STORAGE-7				
2.9	MSTG-STORAGE-9				
2.10	MSTG-STORAGE-10			F	
2.14	MSTG-STORAGE-14	F	F	F	F
Requisitos de criptografia					
3.1	MSTG-CRYPTO-1				F
3.2	MSTG-CRYPTO-2			F	
3.3	MSTG-CRYPTO-3				
3.4	MSTG-CRYPTO-4	F	F	F	F
3.6	MSTG-CRYPTO-6	F	F	F	F
Requisitos de autenticação e gestão de sessão					
4.5	MSTG-AUTH-5				
4.7	MSTG-AUTH-7		F		
4.8	MSTG-AUTH-8		F		
4.9	MSTG-AUTH-9		F		
Requisitos de comunicação em rede					
5.1	MSTG-NETWORK-1		F	F	F
5.2	MSTG-NETWORK-2		F	F	F
5.3	MSTG-NETWORK-3		F	F	
5.4	MSTG-NETWORK-4	F	F	F	F
Requisitos de interacção da plataforma					
6.4	MSTG-PLATFORM-4				
6.5	MSTG-PLATFORM-5		F	F	F
6.6	MSTG-PLATFORM-6				
6.7	MSTG-PLATFORM-7			F	
6.8	MSTG-PLATFORM-8				
6.9	MSTG-PLATFORM-9				
Qualidade de código e requisitos de configuração de construção					
7.2	MSTG-CODE-2		F	F	
Requisitos de resiliência					
8.1	MSTG-RESILIENCE-1		F	F	F
8.2	MSTG-RESILIENCE-2		F	F	F
8.3	MSTG-RESILIENCE-3				
8.4	MSTG-RESILIENCE-4				
8.5	MSTG-RESILIENCE-5		F	F	F
8.6	MSTG-RESILIENCE-6		F	F	F
8.7	MSTG-RESILIENCE-7				

Tabela 5.4: Resultado das aplicações bancárias moey!, Garanti BBVA, ActivoBank e Banco CTT.

5.2.2 Análise de aplicações COVID-19

As aplicações de COVID-19 foram criadas para rastrear a propagação do vírus, mas assegurando o anonimato dos utilizadores. Para este tipo de aplicações é esperado que se cumpram os requisitos L1 + RE. A Tabela 5.5 ilustra os resultados das 4 aplicações analisadas. Optou-se por analisar uma menor quantidade destas aplicações do que as do tipo bancário, devido ao facto da maioria das aplicações COVID-19 usarem a mesma biblioteca, DP-3T [6], para o desenvolvimento do código. Foram escolhidas para análise as aplicações oficiais de Inglaterra, Portugal, França e Rússia.

A aplicação Russa foi a única que impediu a extracção do código em memória, permitindo que esta realize as suas operações como esperado. É também visível na Tabela 5.5 que as aplicações Portuguesa e Inglesa não têm as configurações de SSL corretas, falhando assim os pontos 5.1 e 5.2. O facto destes dois requisitos não estarem implementados pode levar à identificação do utilizador por parte do atacante, incumprindo assim um dos objectivos da criação destas aplicações, que é preservar o anonimato de quem as usar.

#	MSTG-ID	NHS COVID-19 [UK]	Stayaway [POR]	AntiCOVID [FR]	SocMonitoring [RU]
Requisitos de arquitectura, desenho e modelação de ameaças					
1.9	MSTG-ARCH-9				
Armazenamento de dados e requisitos de privacidade					
2.2	MSTG-STORAGE-2	F		F	F
2.3	MSTG-STORAGE-3	F	F	F	F
2.5	MSTG-STORAGE-5				
2.7	MSTG-STORAGE-7				
2.9	MSTG-STORAGE-9				
2.10	MSTG-STORAGE-10			F	
2.14	MSTG-STORAGE-14	F	F	F	F
Requisitos de criptografia					
3.1	MSTG-CRYPTO-1				
3.2	MSTG-CRYPTO-2				
3.3	MSTG-CRYPTO-3				
3.4	MSTG-CRYPTO-4				F
3.6	MSTG-CRYPTO-6	F		F	F
Requisitos de autenticação e gestão de sessão					
4.5	MSTG-AUTH-5				
4.7	MSTG-AUTH-7		F		
4.8	MSTG-AUTH-8		F		
4.9	MSTG-AUTH-9		F		
Requisitos de comunicação em rede					
5.1	MSTG-NETWORK-1	F	F		
5.2	MSTG-NETWORK-2	F	F		
5.3	MSTG-NETWORK-3				F
5.4	MSTG-NETWORK-4	P	P	P	P
Requisitos de interacção da plataforma					
6.4	MSTG-PLATFORM-4				
6.5	MSTG-PLATFORM-5	F			
6.6	MSTG-PLATFORM-6				
6.7	MSTG-PLATFORM-7				
6.8	MSTG-PLATFORM-8				
6.9	MSTG-PLATFORM-9				
Qualidade de código e requisitos de configuração de construção					
7.2	MSTG-CODE-2	F	F	F	F
Requisitos de resiliência					
8.1	MSTG-RESILIENCE-1	F	F	F	P
8.2	MSTG-RESILIENCE-2	F	F	F	
8.3	MSTG-RESILIENCE-3				
8.4	MSTG-RESILIENCE-4				
8.5	MSTG-RESILIENCE-5	F	F	F	
8.6	MSTG-RESILIENCE-6	F	F	F	
8.7	MSTG-RESILIENCE-7				

Tabela 5.5: Resultados da análise do ransAPK sobre as aplicações COVID-19.

5.2.3 Análise de uma aplicação *Packed*

De forma a termos um método de comparação entre o ransAPK e o MobSF, na análise de aplicações com o seu código protegido, escolhemos uma aplicação com esta característica, a KrazyRupee. Os resultados podem ser vistos na Tabela 5.6. Conforme esperado, e uma vez que o MobSF se baseia na análise estática para validar os requisitos de segurança, esta ferramenta mostrou-se incapaz de produzir qualquer resultado. Isto porque para uma aplicação *Packed* obter o seu código, é necessário efectuar uma chamada a um servidor remoto para pedir os ".Dex" que contêm o código original da aplicação. Este é o perfil normal de aplicações que estão *Packed*. Existem outras que colocam os ficheiros ".Dex" dentro dos *Assets* do APK. Esta solução é pouco usada, mas às vezes está disponível, possibilitando a execução da aplicação ainda que o telemóvel não tenha acesso à Internet (recorrendo aos ".Dex" armazenados no localmente).

Para o ransAPK, a situação em que a aplicação está com o código protegido é complemente transparente. O sistema é capaz de executar a sua análise sem impedimentos e detectar se existem requisitos em falta.

#	MSTG-ID	MobSF - KrazyRupee	ransAPK - KrazyRupee
Requisitos de arquitectura, desenho e modelação de ameaças			
1.9	MSTG-ARCH-9		
Armazenamento de dados e requisitos de privacidade			
2.2	MSTG-STORAGE-2		
2.3	MSTG-STORAGE-3		F
2.5	MSTG-STORAGE-5		
2.7	MSTG-STORAGE-7		
2.9	MSTG-STORAGE-9		
2.10	MSTG-STORAGE-10		
2.14	MSTG-STORAGE-14		F
Requisitos de criptografia			
3.1	MSTG-CRYPTO-1		
3.2	MSTG-CRYPTO-2		
3.3	MSTG-CRYPTO-3		
3.4	MSTG-CRYPTO-4		
3.6	MSTG-CRYPTO-6		
Requisitos de autenticação e gestão de sessão			
4.5	MSTG-AUTH-5		
4.7	MSTG-AUTH-7		F
4.8	MSTG-AUTH-8		F
4.9	MSTG-AUTH-9		F
Requisitos de comunicação em rede			
5.1	MSTG-NETWORK-1		F
5.2	MSTG-NETWORK-2		F
5.3	MSTG-NETWORK-3		
5.4	MSTG-NETWORK-4		
Requisitos de interacção da plataforma			
6.4	MSTG-PLATFORM-4		
6.5	MSTG-PLATFORM-5		F
6.6	MSTG-PLATFORM-6		
6.7	MSTG-PLATFORM-7		
6.8	MSTG-PLATFORM-8		
6.9	MSTG-PLATFORM-9		
Qualidade de código e requisitos de configuração de construção			
7.2	MSTG-CODE-2		F
Requisitos de resiliência			
8.1	MSTG-RESILIENCE-1		F
8.2	MSTG-RESILIENCE-2		F
8.3	MSTG-RESILIENCE-3		
8.4	MSTG-RESILIENCE-4		
8.5	MSTG-RESILIENCE-5		F
8.6	MSTG-RESILIENCE-6		F
8.7	MSTG-RESILIENCE-7		

Tabela 5.6: Resultado da análise do ransAPK e MobSF sobre uma aplicação com código protegido.

5.3 Precisão do ransAPK

De forma a avaliar a precisão do ransAPK, iremos nos focar nos requisitos que esta ferramenta detectou, e verificar se estes são verdadeiros ou falsos positivos.

Para o grupo de *Requisitos de resiliência*, sempre que o ransAPK consegue extrair o código e efectuar a análise de código estático, coloca os requisitos 8.1, 8.2, 8.5 e 8.6 como falha (not-pass). Nesta situação não há probabilidade de existir um falso positivo, porque o ransAPK só marca como falha caso haja ficheiros ".Dex" extraídos e convertidos com sucesso para classes Java.

Já nos *Requisitos de comunicação em rede*, conforme foi descrito na Secção SSLabs (Secção 4.3.3), os requisitos falham se a nota retornada para um determinado servidor for inferior a A, garantindo assim que não existirá falsos positivos para os requisitos 5.1 e 5.2.

Os restantes requisitos são analisados por regras Yara, nesta situação existe apenas uma excepção que pode levar a um falso positivo. Particularmente, durante a análise de código estática de um ficheiro ".Dex" não é possível saber se um dado excerto de código será executado.

Nas tabelas que ilustram os resultados, é visível que muitos dos requisitos não têm qualquer conclusão por parte do ransAPK. Isto deve-se ao facto de o ransAPK validar os requisitos com regras que observam métodos de implementação. As regras criadas observam a forma mais comum de cumprir os requisitos, mas o programador pode decidir fazê-lo de outra forma. O ransAPK nestas situações, de maneira a não criar falsos positivos, se uma regra não foi observada, não coloca nenhum resultado. Posteriormente o analista pode validar manualmente e colocar o estado do requisito na interface *web* disponibilizada pelo sistema ransAPK.

Capítulo 6

Conclusão

Nesta dissertação é apresentado o sistema ransAPK para verificação de requisitos de segurança em aplicações *Android*, seguindo as recomendações do *Mobile Security Project* da OWASP. O sistema tem a capacidade de verificar 34 requisitos de forma automatizada, mostrando ao utilizador, via uma interface *web*, quais os requisitos que estão em conformidade, bem como a maturidade da aplicação quanto à implementação de requisitos.

Para validar os requisitos de segurança, o ransAPK recorre às capacidades de análise estática e dinâmica, necessitando de um dispositivo físico ou virtual para a análise da última. Conforme descrito no Capítulo 5, Secção 5.2.4, foram avaliadas 17 aplicações, tendo mostrado que todas elas apresentavam falta de implementação de um número considerável de requisitos de segurança. É de notar que o sistema poderá apresentar falsos positivos, ainda que estes tenham uma baixa probabilidade de ocorrer.

Dos resultados obtidos, é visível que o ransAPK é uma solução mais versátil do que o MobSF, com a capacidade de validar qualquer tipo de aplicação, mesmo tendo o código protegido, isto é *Packed*.

Para trabalho futuro, esta aplicação deve focar-se na transformação dos requisitos avaliados de forma estática em forma dinâmica, uma vez que o sistema apresenta uma fácil integração de novos módulos de verificação de requisitos, e o servidor Frida permite que sejam injectadas instruções enquanto a aplicação se encontra em execução. Contudo, nunca podemos descartar totalmente a análise estática de uma aplicação, porque se esta cumprir os requisitos de resiliência na sua totalidade, não será possível a execução da análise dinâmica. Para além desta melhoria, a interface *web* deve ser melhorada com o intuito de ser mais simples de utilizar e organizar os resultados.

Bibliografia

[1] APKTOOL. Apktool official documentation. https://ibotpeaches.github.io/Apktool/documentation/, 2020.

[2] Yauhen Leanidavich Arnatovich, Lipo Wang, Ngoc Minh Ngo, and Charlie Soh. A comparison of android reverse engineering tools via program behaviors validation based on intermediate languages transformation. *IEEE Access*, 6(c):12382–12394, 2018.

[3] Alexandre Bartel, Jacques Klein, and Martin Monperrus. Dexpler: Converting android dalvik bytecode to jimple for static analysis with soot. *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program Analysis, SOAP 2012*, (May 2014):27–38, 2012.

[4] codifiedsecurity. codifiedsecurity official web-page. https://codifiedsecurity.com/, 2020.

[5] Dex2Jar. Dex2jar official page. https://sourceforge.net/p/dex2jar/wiki/Faq/, 2020.

[6] DP-3T. Dp-3t oficial repository. https://github.com/DP-3T/documents, 2020.

[7] Abhishek Dubey and Anmol Misra. *ANDROID SECURITY ATTACKS AND DEFENSES*.

[8] Nikolay Elenkov. *Android Security Internals*, volume 2015. 2015.

[9] Ana Fernandes, António Ravara, and João Casal. App Threat Analysis: Combining static analysis with users feedback to accelerate app store resposne to mobile threats. 2018.

[10] Ahmad Fikri, Alfan Presekal, Ruki Harwahyu, and Riri Fitri Sari. Performance comparison of dalvik and ART on different android-based mobile devices. *2018 International Seminar on Research of Information Technology and Intelligent Systems, ISRITI 2018*, pages 439–442, 2018.

[11] Frida. Frida official page. https://frida.re/docs/android/, 2020.

[12] Andrei Frumusanu. A closer look at android runtime (art) in android l. https://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l, 2014.

[13] GOOGLE. Adb documentation. https://developer.android.com/studio/command-line/adb?hl=pt-br, 2020.

[14] Google. Malware categories. https://developers.google.com/android/play-protect/phacategories#trojan, 2020. [Online; accessed 1-October-2020].

[15] R. Canetti H. Krawczyk, M. Bellare. Hmac: Keyed-hashing for message authentication. https://tools.ietf.org/html/rfc210, 2020.

[16] hluwa. Dexdump repository. https://github.com/hluwa/FRIDA-DEXDump, 2020.

[17] Chun-ying Huang and Chih-hung Lin. Code Coverage Measurement for Android Dynamic Analysis Tools. *IEEE Software*, 32(2):c2–c2, 2015.

[18] Ionut Ilascu. Major instagram app bug could've given hackers remote access to your phone. https://www.bleepingcomputer.com/news/security/fake-threema-telegram-apps-hide-spyware-for-targeted-attacks/, 2020. [Online; accessed 1-October-2020].

[19] ImmuniWeb. Immuniweb official web-page. https://www.immuniweb.com/mobile, 2020.

[20] ImmuniWeb. Immuniweb official web-page integrations. https://www.immuniweb.com/integrations/, 2020.

[21] ImmuniWeb. Immuniweb official web-page pci dss compliance. https://www.immuniweb.com/compliance/pcidss/, 2020.

[22] ImmuniWeb. Immuniweb official web-page pci gpdr compliance. https://www.immuniweb.com/compliance/gdpr/, 2020.

[23] J.Clement. Distribution of free and paid android apps in the google play store as of september 2020. https://www.statista.com/statistics/266211/distribution-of-free-and-paid-android-apps, 2020.

[24] Daniel Reynaud Jonathan Meyer. Jasmin - a java assembler. https://sourceforge.net/projects/jasmin/, 2020.

[25] Ravie Lakshmanan. Major instagram app bug could've given hackers remote access to your phone. https://thehackernews.com/2020/09/instagram-android-hack.html, 2020. [Online; accessed 18-july-2020].

[26] Cell Culture Manual. OWASP Mobile Application Security Verification Standard. pages 2–4, 2010.

[27] MOBFS. Mobfs repository. https://mobsf.github.io/docs/#/, 2020.

[28] OWASP. Wasp mobile top 10. https://owasp.org/www-project-mobile-top-10/, 2016.

[29] OWASP. OWASP Mobile Security Testing Guide - 1.1.3 Release. page 535, 2019.

[30] owasp. Man-in-the-middle attack. https://owasp.org/www-community/attacks/Man-in-the-middle_attack, 2020.

[31] Kai Qian, Reza M. Parizi, and Dan Lo. OWASP Risk Analysis Driven Security Requirements Specification for Secure Android Mobile Software Development. *DSC 2018 - 2018 IEEE Conference on Dependable and Secure Computing*, pages 1–2, 2019.

[32] Rohit Tamma, Oleg Skulkin, Heather Mahalik, and Satish Bommisetty. *Practical mobile forensics : a hands-on guide to mastering mobile forensics for the iOS, Android, and the Windows phone platforms*. 2018.

[33] SANS. 2020 cwe top 25 most dangerous software weaknesses. https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html, 2020.

[34] P C I Security and Standards Council. The Prioritized Approach to Pursue PCI DSS Compliance. *PCI Security standards council*, (May):1–18, 2016.

[35] Himanshu Shewale, Sameer Patil, Vaibhav Deshmukh, and Pragya Singh. Analysis of Android Vulnerabilities and Modern Exploitation Techniques. *ICTACT Journal on Communication Technology*, 05(01):863–867, 2014.

[36] SOOT. Soot official page. http://soot-oss.github.io/soot/, 2020.

[37] VUEJS. Vuejs official web page. https://vuejs.org/, 2020.

[38] Miao Wei. Art vs dalvik - introducing the new android* x86 runtime. https://software.intel.com/content/www/us/en/develop/blogs/art-vs-dalvik-introducing-the-new-android-x86-runtime.html, 2014.

[39] Radhakishan Yadav and Robin Singh Bhadoria. Performance analysis for android runtime environment. *Proceedings - 2015 5th International Conference on Communication Systems and Network Technologies, CSNT 2015*, pages 1076–1079, 2015.

[40] Yara. Yara oficial documentation. `https://yara.readthedocs.io/en/stable/`, 2020.

[41] Marwa Ziadia, Jaouhar Fattahi, Mohamed Mejri, and Emil Pricop. Smali+: An operational semantics for low-level code generated from reverse engineering android applications. *Information (Switzerland)*, 11(3), 2020.