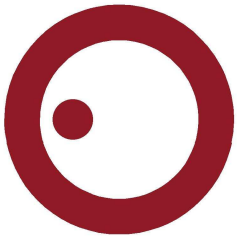


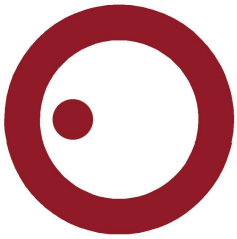
escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

TÉCNICAS DE ANÁLISE DINÂMICA A APLICAÇÕES ANDROID

LUÍS FILIPE CAIXEIRO SANTOS

Leiria, Março de 2024



escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

TÉCNICAS DE ANÁLISE DINÂMICA A APLICAÇÕES ANDROID

LUÍS FILIPE CAIXEIRO SANTOS
Número: 2210850

Projeto realizado sob orientação do Professor Doutor Miguel Monteiro de Sousa Frade e Professor Doutor Patrício Rodrigues Domingues.

Leiria, Março de 2024

AGRADECIMENTOS

Gostaria de começar por agradecer a todos os que tornaram a conclusão deste trabalho possível. Desde logo aos meus dois orientadores Professor Doutor Patrício Rodrigues Domingues e o Professor Doutor Miguel Monteiro de Sousa Frade, aos quais sem eles, este trabalho ainda estaria arrumado na gaveta.

Além deles quero também agradecer a boa disposição e espírito de camaradagem dos meus colegas de mestrado, os quais partilhámos bons e difíceis momentos devido ao horário pós laboral ao qual estávamos sujeitos.

Por último e não menos importante gostaria de agradecer a toda a minha família, pelo apoio e motivação para concluir mais esta etapa.

Não foi fácil aqui chegar e o resultado final nem sempre é o que esperamos alcançar. Como tal ficamos pela frase de Voltaire, "o ótimo é inimigo do bom", que para mim serviu e para outros será uma realidade.

RESUMO

A constante evolução e popularidade dos dispositivos móveis transformaram a forma como comunicamos e agimos nas mais diversas atividades que fazem parte do quotidiano, sejam pessoais ou profissionais. Neste contexto, aplicações para dispositivos móveis emergiram de uma forma exponencial para facilitar e aproveitar o crescente uso deste tipo de dispositivos, transformando-se num sector de negócio em constante crescimento.

A necessidade de garantir a segurança dos seus utilizadores emerge como prioridade, para isso este trabalho procura estudar a metodologia de análise à segurança e ao comportamento de aplicações móveis em ambiente *Android*. Este trabalho propõe uma investigação detalhada recorrendo às metodologias de análise mais apropriadas para averiguar o comportamento de uma aplicação móvel e validar os mecanismos de segurança implementados pela própria aplicação. Para o efeito é efetuado uma análise dinâmica e estática sobre uma aplicação móvel, não esquecendo ainda a procura e análise de artefactos digitais forenses resultantes do uso da aplicação.

Toda a informação que foi obtida resultante das respetivas análises é documentada de forma detalhada, assim como a metodologia utilizada. Do estudo efetuado, foi percetível que a aplicação *Bolt* recolhe informação além do considerado necessário. O excesso de informação recolhida afeta tanto os utilizadores, como o hardware incluindo informação de rede e localização, a qual em muitos casos é não necessária para a função ou depuração da aplicação.

Este trabalho refere com detalhe os diretórios e artefactos com maior valor forense que são recolhidos pela aplicação *Bolt*. Estes artefactos podem ser relevantes, por exemplo, para forças policiais ou judiciais no decurso das respetivas investigações. Os artefactos podem também ser importantes para os utilizadores da aplicação *Bolt* terem conhecimento dos dados recolhidos pela aplicação, assim como dos *trackers* existentes e a informação que é recolhida sem que sejam alertados devidamente para o mesmo.

ABSTRACT

The constant evolution and popularity of mobile devices have transformed the way we communicate and engage in various everyday activities, whether personal or professional. In this context, mobile applications have emerged exponentially to facilitate and capitalize on the growing use of such devices, evolving into a continuously growing business sector.

Ensuring the security of its users emerges as a priority. Thus, this work seeks to study the methodology for analyzing the security and behavior of mobile applications in the Android environment. This study proposes a detailed investigation utilizing the most appropriate analysis methodologies to assess the behavior of a mobile application and validate the security mechanisms implemented by the application itself. This involves both dynamic and static analysis of a mobile application, along with the search and analysis of digital forensic artifacts resulting from the application's use.

All information obtained from these respective analyses is thoroughly documented, along with the methodology used. From the study conducted, it was evident that the 'Bolt' application collects information beyond what is deemed necessary. The excess information collected affects both users and hardware, including network and location information, which in many cases is unnecessary for the application's function or debugging.

This work meticulously outlines the directories and artifacts with significant forensic value collected by the 'Bolt' application. These artifacts may be relevant, for example, to law enforcement or judicial authorities during their respective investigations. Additionally, these artifacts may be crucial for users of the 'Bolt' application to be aware of the data collected by the application, as well as the presence of trackers and information collected without proper notification.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

2FA	Two-Factor Authentication.
ABE	Android backup extractor.
ADB	Android Debug Bridge.
AES	Advanced Encryption Standard.
AiTM	Adversary-in-the-Middle.
AM	Activity Manager.
AOSP	Android Open Source Project.
AOT	Ahead of Time.
API	Application Programming Interface.
APK	Android Application Package.
ARM	Advanced RISC Machine.
ART	Android Runtime.
ASLR	Address Space Layout Randomization.
BD	Base de dados.
BLOB	Binary Large OBject.
CA	Certificate Authority.
CDN	Content Delivery Network.
CE	Credential Encrypted.
CPU	Central Processing Unit.
DAL	Digital Asset Links.
DB	DataBase.
DE	Device Encrypted.

DEP	Data execution prevention.
DEX	Dalvik Executable.
DM	Dispositivo Móvel.
DNS	Domain Name System.
DoT	DNS over TLS.
DVM	Dalvik Virtual Machine.
EAP	Extensible Authentication Protocol.
EBT	Essemble bagged tree.
EID	European Investment Bank.
ELF	Executable and linking format.
Far	False accept rate.
FBE	File-Based Encryption.
FCM	Firebase Cloud Messaging.
FDE	Full-Disk Encryption.
FS	File System.
GCC	Google Cloud Console.
GCM	Galois/Counter Mode.
GMS	Google Mobile Services.
GPS	Global Positioning System.
GUI	Graphical User Interface.
HAL	Hardware Abstraction Layer.
HFS	Mac OS Extended.
HOTP	HMAC-Based One-time Password.
HTTP	Hypertext Transfer Protocol.
IDE	Integrated development environment.

IETF	Internet Engineering Task Force.
IMEI	International Mobile Equipment Identity.
IO	Input/Output.
IoT	Internet of Things.
IP	Internet Protocol.
IPC	Inter-process communication.
JAR	Java Archive.
JCE	Java Cryptographic Extension.
JIT	Just in Time.
JNI	Java Native Interface.
JSON	JavaScript Object Notation.
JVM	Java Virtual Machine.
JWT	Json Web Token.
KNN	k-Nearest neighbors.
LMKD	Low Memory Killer Daemon.
LRU	Least Recently Used.
LTS	Long-term support.
MAC	Media access control address.
MAS	Mobile Application Security.
MASTG	Mobile Application Security Testing Guide.
MASVS	Mobile Application Security Verification Standard.
MD	Mobile Devices.
MF	Mobile Forensics.
MITM	Man-In-The-Middle.
NDK	Native Development Kit.

NFC	Near Field Communication.
OEM	Original Equipment Manufacturer.
OS	Operative System.
OTP	One-Time Password.
OWE	Opportunistic Wireless Encryption.
PGO	Profile Guided Optimization.
PID	Process Identificator.
PII	Personally Identifiable Information.
POC	Proof of Concept.
PSK	Pre-shared key.
QUIC	Quick UDP Internet Connections.
RAM	Random Access Memory.
REPL	Read evaluate print loop.
RFC	Request for Comments.
RGPD	Regulamento Geral sobre a Proteção de Dados.
RHS	Ride-Hailing Service.
RIB	Router, Interactor and Builder.
ROM	Read Only Memory.
SAE	Simultaneous Authentication of Equals.
SD	Secure Digital Card.
SDK	Software Development Kit.
SE	Secure Element.
SMS	Short Message Service.
SO	Sistema Operativo.
SQL	Structured Query Language.

SSH	Secure Shell.
SSL	Secure Sockets Layer.
SVM	Support vector machine.
TAR	True accept rate.
TCP	Transmission Control Protocol.
TEE	Trusted Execution Environment.
TLS	Transport Layer Security.
TOTP	Time-based One-time Password.
TWRP	Team Win Recovery Project.
UDP	User Datagram Protocol.
UID	User Identification.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
VM	Virtual Machine.
VoIP	Voice over Internet Protocol.
VPN	Virtual Private Network.
WFA	Wi-Fi Alliance.
WPA	Wi-Fi Protected Access.
XML	Extensible Markup Language.
XSS	Cross-site Scripting.

1

INTRODUÇÃO

Existem mais de 3 mil milhões de dispositivos móveis que fazem uso do Sistema Operativo (SO) Android, representando 75%[1] de todos os *smartphones* em uso a nível mundial, prevendo-se que seja alcançado a quantidade de 4 mil milhões de dispositivos em uso no ano 2025. Os *smartphones* tornaram-se ferramentas essenciais na vida moderna, armazenando uma vasta quantidade de informações pessoais e profissionais.

Muitos atos do dia-a-dia têm lugar no *smartphone*, sejam compras, acesso à banca online, jogos, redes sociais, sistemas vestíveis, realidade aumentada, consulta de informação, entre muitas outras atividades. Não surpreende pois que o *smartphone* seja um repositório de dados pessoais e profissionais.

As aplicações móveis são elementos essenciais do ecossistema *smartphones*, potenciando as funcionalidades dos dispositivos. A comprovar a importância das aplicações móveis está o volume de download de aplicações, que em 2023 se situou na ordem dos 257 mil milhões de downloads[2]. Muitas das aplicações recolhem dados dos utilizadores sem que os mesmos tenham plena consciência da situação dado as políticas de utilização, para as quais o utilizador deve consentir, serem usualmente redigidas com um vocabulário e um texto denso, que nem sempre é fácil de interpretar.

Uma das funcionalidades tornadas possíveis com os *smartphones* foram as aplicações de serviço de transporte, em que o utilizador solicita, através de uma aplicação específica, um transporte de um ponto A até a um ponto B. Este tipo de serviço possibilita ao utilizador o solicitar transporte de modo flexível, tendo uma estimativa do preço de forma antecipada e do tempo previsível de espera, permitindo ainda efetuar o pagamento de forma eletrónica e sem interação direta entre o utilizador e o motorista. As aplicações de serviço de transporte requerem um elevado nível de segurança, dado que recolhem e processam dados sensíveis tais como a geolocalização

1 Statista - Quota de mercado de sistemas operativos móveis - Disponível em: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ - Acedido a 22 Mar. de 2024

2 Statista - Número de downloads anuais entre 2016-2023 para aplicações móveis- Disponível em: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/ - Acedido a 22 Mar. de 2024

dos utilizadores e motoristas/veículos afetos ao serviço de transporte, bem como os meios de pagamento – cartão de crédito/débito dos utilizadores do serviço. A *Bolt: Request a Ride* é precisamente uma dessas aplicações que providencia serviço de transporte, tendo no momento de redação deste documento mais de 50 milhões de downloads no *Google Play* e mais de 5,8 milhões de avaliações, com um resultado médio de 4.7 em 5[3]. Pelo facto de se tratar de uma aplicação exigente do ponto de vista de segurança, com um elevado número de utilizadores, e com uma avaliação elevada por parte dos mesmos, foi selecionada como caso de estudo para este trabalho.

1.1 MOTIVAÇÃO

Tendo em conta que as aplicações móveis podem manipular dados pessoais, e que são executadas no dispositivo móvel, é de todo importante determinar se uma dada aplicação é segura e confiável. De facto, uma aplicação insegura pode perigar todo o *smartphone*, potencialmente expondo o utilizador a roubo de dados, de recursos, a falsa informação, etc. Por sua vez, uma aplicação não confiável poderá ela própria exfiltrar dados do utilizador, ou aproveitar-se dos recursos do dispositivo.

A motivação principal deste trabalho é o estudo da metodologia para análise à segurança e ao comportamento de aplicações móveis em ambiente *Android*. Pretende-se determinar as metodologias de análises mais apropriadas para averiguar o comportamento de uma aplicação móvel e validar os mecanismos de segurança implementados pela própria aplicação.

Para o efeito é efetuado o estudo da aplicação móvel *Bolt* para *Android*, recorrendo-se a um vasto leque de métodos de análise estática e análise dinâmica usualmente empregues na análise de aplicações móveis. Este trabalho incide ainda na procura e análise de vestígios digitais resultantes da execução da aplicação *Bolt*, expondo assim os artefactos digitais forenses. Assim, muito da metodologia de análise empregue é coincidente com a usada na informática forense. É ainda dado grande ênfase à documentação das metodologias empregues, recorrendo-se sempre que se considere pertinente, a exemplos para ilustração do uso das ferramentas empregues.

3 Play store, Bolt: Request a Ride - Disponível em: https://play.google.com/store/apps/details?id=ee.mtakso.client&hl=pt_PT&gl=US - Acedido a 22 Mar. de 2024

1.2 OBJETIVOS

O principal objetivo do trabalho é realizar uma análise abrangente da segurança e comportamento de aplicações *Android*, procurando compreender todo o processo e metodologias necessárias. Para o efeito é estudada a aplicação *Bolt*.

Um objetivo paralelo é o de documentar, de forma detalhada, os passos e as ferramentas relevantes para identificar, avaliar e mitigar riscos de segurança nas aplicações *Android*. De facto, uma análise à segurança e comportamento a uma aplicação *Android* é um processo relativamente complexo, que requer um vasto leque de conhecimentos e o uso de um número elevado de ferramentas. Pretende-se com este trabalho desmistificar o processo de análise, fornecendo documentação descritiva da metodologia e das muitas ferramentas existentes para o efeito.

1.3 CONTRIBUTOS

De um modo geral, este trabalho pretende contribuir para um melhor entendimento da segurança em aplicações *Android*. Em termos concretos, destacam-se os seguintes contributos:

- Uso e documentação de ferramentas e metodologias de análise estática a uma aplicação móvel *Android* para validação da segurança e do comportamento da aplicação;
- Uso e documentação de ferramentas e metodologias de análise dinâmica a uma aplicação móvel *Android*;
- Aplicação das metodologias estática e dinâmica para a análise da segurança e do comportamento de uma aplicação *Android* com um elevado número de utilizadores, detalhando os respetivos artefactos forenses;
- Descrição detalhada das técnicas e ferramentas utilizadas, combinada com exemplo concretos com o intuito de ilustrar as referidas técnicas.

1.4 ESTRUTURA DO DOCUMENTO

Este documento está organizado da seguinte forma. No Capítulo 2 é apresentada uma revisão do trabalho relacionado, contextualizando o tema abordado. Ainda no mesmo capítulo é apresenta a aplicação *Bolt* e a sua posição de mercado.

O Capítulo 3 explora os conceitos e tecnologias essenciais associados ao ambiente *Android*, incluindo a arquitetura e as camadas mais relevantes. A segurança do sistema operativo *Android* é discutida nas Secções 3.4 e 3.5, enquanto a Secção 3.6 aborda a segurança de rede. Os componentes e a estrutura das aplicações *Android* são analisadas nas Secções 3.9 e 3.10, respetivamente.

No Capítulo 4 é apresentado o ambiente de testes, detalhando-se o processo de *rooting* do *smartphone Android* empregue e as principais vantagens e desvantagens do seu uso. As ferramentas empregues neste trabalho para análise estática e dinâmica a aplicações são apresentadas nas Secções 4.2 e 4.3.

O Capítulo 5 detalha a análise estática à aplicação em estudo, ao passo que o Capítulo 6 documenta a metodologia e os principais artefactos forenses da aplicação *Android*. O Capítulo 7 foca a análise dinâmica da aplicação.

Por fim, o Capítulo 8 apresenta as principais conclusões do trabalho e esboça possível trabalho futuro.

2

TRABALHO RELACIONADO

Este capítulo apresenta conceitos importantes que estão diretamente ligados a técnicas de análise a Dispositivo Móvel (DM) e ao SO *Android*. Vai ser dado um maior enfoque a trabalhos que destaquem vulnerabilidades de segurança, bem como ferramentas de análise dinâmica e/ou estática utilizadas. O objetivo principal foi obter um resumo dos trabalhos existentes mais relevantes em aplicações de *ride sharing*. Este capítulo serve também para a introdução de conceitos importantes que estão relacionados com o tema desta dissertação, que nos ajudem a definir o seu escopo e âmbito. É assim importante definir as diferenças existentes entre uma análise forense e uma análise dinâmica, esta última ligada ao *pentest* de aplicações móveis a dispositivos *Android*.

2.1 ENQUADRAMENTO

2.1.1 *Ride Sharing Apps*

O mercado de aplicações de *ridesharing* em 2021 apresentava um valor aproximado de 84.30 mil milhões de dólares Americanos, prevendo-se uma subida para 242.73 mil milhões até 2028, o que representa um crescimento de 16.3% desde 2021 a 2028[1]. Não podemos esquecer que este mercado era praticamente inexistente há uma década atrás.

A sua rápida adoção foi alcançada pela boa acessibilidade e simplicidade de uso das suas aplicações móveis, recorrendo a um *smartphone iOS* ou *Android*, o qual permite planear e agendar a sua viagem, independente da sua localização geográfica. O mercado é composto por diferentes empresas líderes do setor que competem entre si. *Uber e Lyft* competem nos Estados unidos; *Bolt e Uber* na Europa; *Grab e Go-Jek* no Sudeste da Ásia; *Didi e Uber* na Austrália; *Careem e Uber* no Médio

1 Fortune Business Insights - Disponível em: https://www.globenewswire.com/en/news-release/2022/09/26/2522406/0/en/Ride-Sharing-Market-Size-to-Worth-Around-USD-242-73-Billion-by-2028.html - Acedido a 26 de Out. 2022

Oriente; *Ola e Uber* na Índia; e *Cabify, 99 Taxi e Uber* no Brasil (Wang e H. Yang, 2019).

Algumas das aplicações disponibilizam informação adicional com o intuito de reforçar a segurança do utilizador. A informação adicional pode incluir nome, fotografia, identificação do veículo, detalhes da rota e outras informações como histórico de viagens.

2.1.2 *Aplicações Bolt*

A empresa Bolt dispõe de diversos apoios e investimentos com base no seu potencial económico, o qual já atingiu 7.4€ mil milhões de Euros, com base no artigo publicado (EIA, 2022). Parte do apoio chegou também de instituições públicas como o banco europeu de investimento, para desenvolvimento e expansão da tecnologia de transporte de passageiros (*ride-hailing*) e para serviços de entrega de comida. O valor acordado foi de 50 milhões de euros, e é suportado pelo Fundo europeu para investimento estratégico (Tallinn, 2020).

Existem diversas aplicações produzidas pela empresa BOLT, em que o seu principal objetivo é utilizar veículos privados para múltiplas tarefas. Das tarefas fazem parte a requisição de transporte de passageiros (*ride-hailing*), partilha de transporte (*car sharing*), seja de carros ou de *scooters*, distribuição de comida ou entrega de produtos de mercearias. Estas aplicações têm o objetivo de facilitar as ditas tarefas no planeamento da logística necessária para cumprir com os objetivos descritos.

Existem cinco aplicações associadas à empresa *Bolt Technology* que prestam serviço de transporte de diferentes tipos de mercadorias e/ou passageiros.

- Bolt: Viagens & Trotinetes
- Bolt Food
- Bolt Driver: Conduza & Ganhe
- Bolt Courier
- Bolt Restaurant

2.2 ANÁLISE FORENSE

Uma aplicação no contexto de *ride sharing* pode fornecer artefactos com utilidade forense, se a mesma revelar o posicionamento do seu utilizador, com detalhe na data e hora dessa mesma localização. Uma aplicação tem também associada informação privada, como detalhes da conta utilizada na transação monetária utilizada para pagamento da(s) viagem(ns).

A localização é especialmente relevante no contexto forense, para posicionar um possível suspeito no local e/ou data do crime, é também relevante no caso de desaparecimento de uma pessoa perdida, bem como outros casos similares. A esta informação sobre o seu utilizador deve também ser possível identificar o condutor que efetuou a viagem, com elementos como o nome e a foto, de forma a complementar a informação disponível.

Informação detalhada sobre o condutor como referido anteriormente é de extra relevância para uma eventual análise forense. Assim como noutras empresas do mesmo setor, como a *Uber, Lyft e Bolt*, existem casos de agressões sexuais e assaltos cometidos por motoristas nas viaturas de serviço.o (Jacobs, 2022).

Existem também casos relatados por clientes de "falsos"condutores, em casos de indivíduos com antecedentes criminais os quais utilizavam identidades falsas, para serem autorizados a conduzir ou em alguns casos para cometerem crimes. Todos estes casos levaram à evolução de novos mecanismos de identificação do condutor Gupta et al. (2019), por forma a proteger o passageiro e para que exista confiança no sistema em prática.

2.2.1 *Investigação forense de dispositivos móveis*

A investigação forense de dispositivos móveis é por si mesmo um subproduto da investigação forense digital, que por sua vez é apenas uma das ciências forenses. Apesar de todas as ciências forenses terem os mesmo passos no processo de investigação, existem diferenças significativas na investigação forense digital (Kessler, 2015).

Um dos modelos mais completos e competentes utilizado no processo forense é apresentado por (Casey, 2011):

- Identificação;

- Preservação;
- Aquisição;
- Investigação/Procura (*Examination*);
- Análise;
- Relatório e Apresentação;

Uma vez que o objeto de estudo encontra-se em suporte digital, todos os procedimentos e métodos empregues na análise forense digital utilizam como base a ciência para encontrar padrões que suportem a evidência digital encontrada (Kessler, 2015).

Um dos princípios fundamentais de uma análise forense, que é valida para qualquer ramo das ciências forenses é a necessidade de ser mantida no seu estado original qualquer parte da evidência digital. Esta característica é especialmente difícil de cumprir, com as ferramentas forenses a exigirem uma forma de comunicação com o DM. As ferramentas forenses tradicionais não protegem com o bloqueio de escrita durante uma aquisição forense, comprometendo assim a validade da sua análise.

Para que estes sejam válidos é necessário que os resultados obtidos possam ser reproduzidos e replicados, por diferentes analistas em ambientes laboratoriais. No entanto a interpretação aos resultados obtidos pode em alguns casos gerar diferentes conclusões que podem ser influenciados por fatores externos/culturais/legislativos, entre outros.

2.2.2 *Análise forense de dispositivos móveis*

Análise forense de dispositivos móveis é um ramo da ciência digital forense, a qual envolve o processo de identificação, aquisição, análise e preservação de qualquer evidência que foi obtida de um dispositivo móvel (Reddy, 2019).

Devido ao uso extensivo de dispositivos móveis, estes podem ser objeto sobre o qual recaiam crimes de cibersegurança. Isto deve-se em parte à sua própria portabilidade, que o atribui um valor de objeto pessoal, o qual é usado como meio de identificação do seu portador e arquivo de dados os quais podem ser em diferentes momentos pessoais e/ou comerciais, aliada à facilidade de realizar operações de transações online.

A análise forense de dispositivos móveis apresenta grandes diferenças em relação à análise clássica de computadores. A existência de diferentes arquiteturas de *CPU*, marcas e modelos de dispositivos com um conjunto diferenciado de SO's com

diferentes implementações e versões de segurança (ex. *API lvl* - no contexto de Android), todas estas características contribuem para tornar o processo forense mais complexo. O método forense utilizado irá assim depender do equipamento e da versão do SO, podendo o mesmo método não funcionar num equipamento mais antigo (Barmpatsalou, Damopoulos et al., 2013).

Tabela 1: Limitações encontradas em investigações a DM.

Fonte: (Barmpatsalou, Cruz et al., 2018)

Limitação	**Descrição**
Manipulação de Dados	*Anonymity-enforced browsing* e serviços de anonimização; Tamanho considerável de dados recolhidos durante investigação;
Ferramentas Forenses	Abordagem forense foca-se demasiado em técnicas de aquisição, colocando em segundo plano o processo de investigação;
Diversidade de SO e dispositivos	As diferentes tecnologias, aumentam a divergência entre as ferramentas Forenses, tanto em termos de funcionalidades como demonstração de resultados. (Votipka et al., 2013)
Aspetos de Segurança	Existe um maior cuidado e sofisticação na criação de métodos anti-forenses.
Cloud	Ferramentas de Mobile Forensics (MF) não contemplam aspetos inerentes numa investigação na *Cloud* tais como, o devido acesso a dados forenses de acordo com os diferentes sistemas judiciais legais envolvidos e a segurança forense dos dados adquiridos.
Processo de automatização	Ainda não está definido se o processo de investigação deverá respeitar um modelo específico para cada dispositivo, ou se este deve ser genérico de forma a ser instituído um *standard* através de um conjunto de linhas orientadoras devidamente aplicáveis a procedimentos forenses (Farjamfar e Abdullah, 2014).

A Tabela 1, apresenta um conjunto de limitações que são encontradas por investigadores forenses no processo de investigação.

Devido às medidas de segurança existentes inerentes aos DM modernos é por vezes necessário ao investigador/analista, quebrar as medidas de segurança (*break in*) através de um *exploit* o qual irá em certa parte alterar os dados presentes no dispositivo. Esta ação por si só pode gerar uma violação legal e gerar problemas num contexto de investigação forense.

Uma das maiores dificuldades forenses neste ramo móvel advém da sincronização de dados com diversos dispositivos. Sendo esses mesmos dados voláteis podem ser facilmente alterados ou removidos de forma remota, o que exige um maior cuidado na preservação dos dados.

Este tipo de equipamentos exige também uma maior atenção ao nível de bateria e o seu consumo, de forma a se garantir que os dados voláteis não sejam perdidos. No entanto outras preocupações devem ser tidas em conta, no caso de o dispositivo estar ligado mas bloqueado por um código/chave pode ser remotamente instruído

pelo criminoso a apagar todo o seu conteúdo. Pode assim existir a necessidade de uso de uma caixa *Faraday* para bloquear todas as comunicações remotas com o mesmo, isto se não for possível desligar todas as comunicações do mesmo de outra forma.

Dispositivos móveis são sistemas dinâmicos que introduzem muitas dificuldades durante a extração e subsequente análise. Com o seu constante aumento associado a uma diversidade de contextos e aplicações, torna difícil a criação de uma ferramenta ou procedimento transversal a todos estes (Tamma et al., 2020).

A Tabela 2 descreve de forma não exaustiva, as dificuldades encontradas por um analista neste ramo das ciências forenses.

2.3 TRABALHOS RELACIONADOS

A análise forense a dispositivos móveis apesar de ser uma área relativamente madura continua a apresentar um conjunto de desafios que necessitam de ser resolvidos. Os desafios que se mantém derivam da grande complexidade e multiplicidade de novos dispositivos que surgiram ao longo da ultima década. Além disso as aplicações que são parte integrante dos dispositivos móveis têm, também estas inovações e alterações de comportamento que obrigam os analistas a acompanhar e a desenvolver ferramentas forenses que ajudem no processo de aquisição de evidência.

O artigo desenvolvido por Casino et al. (2022), que estuda as diversas limitações e desafios emergentes numa análise forense digital, apresenta de forma discriminada os seus resultados para cada domínio das ciências forenses. Para dispositivos móveis, existem desafios que são encontrados frequentemente por profissionais e investigadores nesta área.

A maior limitação apontada é a necessidade de formas de aquisição confiáveis independentemente da sua origem e a sua devida interpretação. Ao qual se junta a necessidade de existirem processos de monitorização do diverso hardware dos dispositivos, que recolha de forma dinâmica os dados e *logs* gerados. Esta dificuldade é transversal aos diversos domínios da ciência digital forense *(Cloud, Network, Mobile, Internet of Things (IoT), File System (FS) & DataBase (DB) e Multimédia)*. A fragmentação de dados é um dos fatores que mais dificulta uma investigação, esta é uma das limitações sentidas no processo de aquisição de dados.

A segunda maior limitação encontrada de acordo com Casino et al. (2022), advém da utilização de métodos e estratégias anti-forenses, que impedem ou dificultam os investigadores e as ferramentas empregues de atingir o seu objetivo.

Existem diversos exemplos de metodologias anti-forense Conlan et al. (2016), utilizadas em dispositivos móveis, desde encriptação, ofuscação de dados, eliminação de artefactos, esteganografia e alteração de imagem Qureshi e El-Alfy (2019), comunicações encriptadas/protegidas (*tunneling, onion routing*), *malware* que se apercebe de técnicas de *sandbox/debug* Guibernau (2020), Virtual Machine (VM) P. Chen et al. (2016) e métodos/técnicas de contra análise.

No estudo apresentado por Horsman (2017), conseguimos perceber de que forma as técnicas anti-forenses limitam a capacidade dos investigadores, para descrever a problemática encontrada com base na realidade do Reino unido, refere que 60% dos casos foram dados como "não passíveis de acusação", em casos que envolviam dados encriptados.

Tal como referido por Harris (2006), métodos anti-forenses exploram a dependência/necessidade de incluir elementos humanos em ferramentas forenses, bem como a limitação existente no hardware em termos de arquitetura e poder computacional. Por forma a ultrapassar esta limitação é necessário um bom treino e conhecimento por parte de investigadores/profissionais por forma a tomar os melhores procedimentos forenses de forma a minimizar o impacto dos métodos e técnicas anti-forenses (Qureshi e El-Alfy, 2019).

2.3.1 *Study of identifying and managing the potential evidence for effective Android forensics*

No trabalho *Study of identifying and managing the potential evidence for effective Android forensics*, os autores Kim e Lee (2020) dão a perceber a importância dos dispositivos móveis para uma análise forense digital. Estes são responsáveis por manter num mesmo dispositivo informação tanto pessoal como profissional, o que os torna um peça fundamental na procura por evidência(s).

Este trabalho transmite ao leitor as dificuldades encontradas por investigadores na área forense, nomeadamente a multiplicidade de aplicações existentes, com diferentes comportamentos, muitas vezes ainda desconhecidas do ponto de vista forense e em constante alteração. Cada aplicação guarda de forma diferente (estrutura) a informação que é gerada pela mesma, o que pode originar a necessidade de

atualização da ferramenta forense no caso de se tratar de uma solução comercial, ou através da manipulação manual.

Esta característica torna o processo moroso e difícil de analisar aplicações com as ferramentas comuns ao seu dispor, o que leva a que seja necessário muitas vezes analisar a aplicação de forma manual.

O analista poderá inadvertidamente analisar informação pessoal presente no dispositivo o que poderá infringir a privacidade do seu utilizador e ao mesmo tempo poderá não conseguir obter a informação para constituir a evidência necessária.

Os autores do artigo criaram uma ferramenta forense para *Android*, a qual recebe um ficheiro de uma imagem *Android* e gera um ficheiro `AFXML` como output, que consiste em informação de uma potencial evidência. No artigo está referido o procedimento de funcionamento da aplicação desenvolvida e os casos de usos possíveis para se realizar investigações a dispositivos móveis com o SO *Android*.

2.3.2 *Geo-locating Drivers: A Study of Sensitive Data Leakage in Ride-Hailing Services*

O estudo conclui que de 20 diferentes aplicações de *ride sharing*, as quais incluem as maior do mercado a Uber, Lyft foi possível recolher dados sobre os seus condutores de todas as 20 aplicações em estudo (Zhao et al., 2019). A função que permite obter esta informação é conhecida como *nearby cars*, tal como o nome indica esta função é destinada para perceber se existem viaturas da empresa em questão nas redondezas.

No entanto não só é possível obter a informação sobre os condutores nas imediações, mas também referem, que foi possível identificar com alto nível precisão informação privada/sensível o que inclui as moradas mais frequentemente visitadas (residência do condutor) e hábitos de condução.

Para possíveis atacantes é também possível inferir informação como desempenho do condutor, compreender o funcionamento de negócio através de informação como número de viagens efetuadas, utilização das viaturas e presença territorial.

2.3.3 *Digital Forensics for Mobility as a service platform: Analysis of Uber application on Iphone and Cloud*

Este trabalho Matulis e Karabiyik (2022) avalia a qualidade da informação que é recolhida pela aplicação da Uber, devido ao seu valor forense é importante determinar onde grande parte da informação reside, seja esta na *Cloud* ou no DM. Para atingir este objetivo utilizaram um dispositivo *iPhone* e recolheram imagens forenses de três formas distintas de aquisição. A Tabela 3 demonstra o software forense empregue e o objetivo na recolha de cada uma das aquisições.

Na discussão de resultados obtidos ficou demonstrado que a ferramenta `Magnet AXIOM` conseguiu extrair uma maior número de localizações oito para quatro da `Cellebrite UFED 4PC`.

Quanto à localização da informação ficou também demonstrado que é possível obter mais informação a partir da *Cloud*, em relação à aquisição lógica, o mesmo foi verdadeiro em relação ao nível de detalhe da mesma informação. Com esta análise forense foi possível averiguar que informação pessoal passível de identificação, bem como dados de localização podem ser obtidos pela Uber, estando estas presentes no dispositivo móvel (locais) e na *Cloud*.

Esta informação no parecer do autor desta dissertação é relevante, uma vez que na maior empresa do segmento é possível extrair informação, tanto a partir da *Cloud* como do DM e a informação disponível é considerada privada/sensível. Podemos assim inferir que a mesma prática poderá ocorrer com grande grau de certeza noutras aplicações de empresas do mesmo setor.

2.3.4 *DriverAuth: A risk-based multi-modal biometric-based driver authentication scheme for ride sharing platforms*

O estudo realizado por Gupta et al. (2019), foi criado devido à necessidade de melhorar a segurança dos passageiros durante a viagem, assegurando que o condutor da viatura seja devidamente autenticado. Este trabalho descreve um modelo de autenticação designado por *DriverAuth*, que utiliza três modelos de biometria (face, voz e toque) para autenticação dos condutores. Como vimos anteriormente em outros estudos a criação de perfis falsos para condutores põem em risco a segurança dos clientes e este serviço (transporte de passageiros), pode ser utilizado como veículo para diversos tipos de crimes.

Os autores realizaram testes ao modelo criado, para isso utilizaram um conjunto de amostragem de 10,320 entradas de 86 utilizadores. Foram utilizados os modelos de classificação Support vector machine (SVM), Essemble bagged tree (EBT) e k-Nearest neighbors (KNN). Os quais alcançaram uma média de True accept rate (TAR) de 96.48%, que representa a probabilidade do sistema aceitar corretamente um pessoa autorizada, e False accept rate (Far) de 0.02%, esta é a probabilidade do sistema autorizar incorretamente uma pessoa não autorizada.

Sendo este um modelo de autenticação, os criadores não impõem limitações na sua implementação, contando apenas que este siga o esquema tradicional de autenticação cliente-servidor. No qual é necessário registar o condutor previamente com os seus dados biométricos, e nas comunicações posteriores aceder a uma Base de dados (BD) central que contenha a informação que valide o condutor através dos dados de biometria anteriormente referidos (voz, face e toque). O objetivo deste trabalho é garantir que haja uma maior confiança por parte dos clientes ao utilizar este serviço como meio de transporte.

2.4 TÉCNICAS DE ANÁLISE

É necessário recolher informação decorrente do uso de aplicações móveis, de forma a se perceber de que forma estas são empregues. Esta recolha de informação também conhecida por *mobile analytics* deve ser utilizada por analistas de segurança para complementar as técnicas discutidas neste trabalho.

Esta é uma das formas de avaliar e identificar de que forma as suas aplicações estão a ser usadas. Esta analítica nem sempre está disponível em todas as aplicações, por vezes a capacidade financeira de manter uma equipa dedicada à segurança pode ditar a criação e avaliação ou não destes indicadores de segurança.

Existem no entanto vários métodos empregues pelos analistas de segurança para avaliar possíveis vulnerabilidades de uma aplicação móvel. Estes métodos incluem análise estática, análise dinâmica, análise do lado do servidor (*server-side*), análise de rede e análise híbrida (ÖZGÜR et al., 2021).

Os métodos de análise referidos tem que ser executados em separado pelo analista, o que torna este processo laborioso e complexo, situação que se torna ainda mais complexa aliada ao facto de não existir um *standard* definido.

2.4.1 *Análise Estática*

De uma forma resumida, a análise estática pressupõe compreender o funcionamento da aplicação sem executar a mesma. Durante uma análise estática é necessário analisar todo o código fonte com o objetivo de compreender a lógica da aplicação, dessa forma podemos anteceder qual o comportamento esperado. Ao perceber o funcionamento é necessário estar atento em especial a violações de privacidade (ex: acesso a informação pessoal do seu utilizador). Existem diversos estudos que recorrem à análise estática para tentar identificar comportamentos maliciosos ou violações de privacidade (Amalfitano et al., 2015; Arzt et al., 2014; Au et al., 2012; Avdiienko et al., 2015; Cao et al., 2015; Chess e Mcgraw, 2004; Christodorescu et al., 2005; Mihai Christodorescu e Jha, 2006; Zimmeck et al., 2017).

No entanto a análise estática por si só não consegue reproduzir uma violação de privacidade, apenas é possível inferir que uma violação pode ocorrer ao executar um determinado código durante a sua execução. Este tipo de análise está sujeita a um maior número de falsos positivos, devido à sua natureza analítica com base em comportamento hipotético e uma análise lógica de alto nível.

Assim as desvantagens da análise estática são:

- O analista tem que deduzir o fluxo dos dados e o seu comportamento através da análise de código.
- A análise ao código pode ser dificultada devido ao uso generalizado de obfuscação do código da aplicação.
- As ferramentas que automatizam este tipo de análise, por norma servem para detetar *malware*, e geram muitos falsos positivos ou identificam problemas que não são relevantes.
- Outro aspeto que a torna um abordagem incompleta, prende-se com a falha em detetar problemas que sejam despoletados através de uma sequência de eventos, dependências externas aos quais se incluiu o estado da própria máquina.

A maior vantagem da análise estática é a facilidade em automatizar ou escalar a análise para multiplicas aplicações. No entanto existem diversos métodos que dificultam o seu uso, tais como a ofuscação de código, ou através da alteração do fluxo de operação da aplicação para esconder a sua forma de funcionamento (Continella et al., 2017; Faruki et al., 2014; Maiorca et al., 2015).

Estudos demonstram que cerca de 30% das aplicações geram ou alteram o seu código em *runtime* em vez de compilar o código existente Lindorfer et al. (2014). Isto

pode acontecer devido a um grande conjunto de condições, seja porque o código que é gerado com base no *input* inserido pelo utilizador ou para permitir uma resposta mais específica ao pedido do seu utilizador. Devido às diversas limitações que foram referidas anteriormente, tendo como objetivo a analise de uma aplicação móvel, não é possível à análise estática contextualizar o comportamento da aplicação. Não querendo com isto dizer que este tipo de análise é irrelevante, mas sobretudo que é necessário complementar com a ajuda da análise dinâmica de forma a confirmar e/ou complementar as vulnerabilidades ou violações de dados encontradas.

Ofuscação de Código

Técnicas de ofuscação de código, as quais incluem não apenas técnicas clássicas que são adaptadas para *Android*, mas também ofuscação de métodos específicos para a plataforma *Android*. O seu uso pode ser visto tanto como uma vantagem, como desvantagem uma vez que é usado por programadores para proteger o seu próprio código contra alteração por parte de atores maliciosos. No entanto pode também este ser usado para esconder o uso de *malware* de forma a evitar a sua deteção por parte de software contra *malware*. Existem trabalhos que referem de que forma essas técnicas afetam e/ou beneficiam os responsáveis pela sua criação, e em concreto de que forma estas afetam do ponto de vista forense (Zhang et al., 2021).

2.4.2 *Análise Dinâmica*

A análise dinâmica por sua vez pretende colmatar as desvantagens apresentadas pela análise estática, ao analisar o comportamento da aplicação durante a sua execução.

Independentemente das técnicas de ofuscação para impedir a análise estática, nenhuma aplicação consegue evitar a sua instrumentalização, uma vez que esta interage com a *framework Android* e executa instruções no seu espaço de memória.

Uma análise dinâmica é desempenhada numa ambiente de testes controlado, o qual compreende um SO alterado ou modificado, de forma ser facilmente observado o comportamento da aplicação (Bhoraskar et al., 2014; S. Chen et al., 2016; Gibler et al., 2012; Liu et al., 2016; Song e Hengartner, 2015; Y. Yang et al., 2013).

Existem diferentes formas de realizar uma análise dinâmica, uma destas é conhecida por *Taint analysis* Arzt et al. (2014) e Min et al. (2019). O seu objetivo é analisar o fluxo de dados durante a execução da aplicação de forma a identificar possíveis fugas de dados sensíveis e operações que sejam executadas sobre estes. Este tipo de análise é também suscetível a ataques de controlo de fluxo Sarwar et al.

(2013) e Schwartz et al. (2010), em que basicamente os atores maliciosos tentam manipular o controlo de fluxo de forma a contornar os mecanismos de segurança existentes na aplicação.

Ações a desempenhar numa análise dinâmica:

- Analisar aspetos específicos de uma aplicação em execução.
- Observar o comportamento dinâmico da aplicação e determinar a função de cada pedaço de código.
- Comportamento passível de analise:
  - Registos de `Logs`
    - Eventos do sistema
    - `Logs` explícitos produzidos pela aplicação e/ou por componentes do sistema.
    - Logs implícitos nos quais exceções produzidas resultantes da execução revelam detalhes sobre o erro ocorrido. Podem ser usados para detetar fugas de dados.
    - $`adb logcat`, ferramenta de linha de comandos útil para obter mensagens de `log` do sistema. A aplicação `Pidcat`[2] produz um resultado idêntico, no entanto é mais fácil de ler e de uso intuitivo.
  - *Intents* enviados ou recebidos
  - Mensagens trocadas entre servidores externos (*REST APIs, Web Sockets*)
  - Conteúdos em memória
  - Ficheiros acedidos e/ou criados
  - Instrumentação de código: chamadas a métodos e à *API* do *Android*

Numa análise dinâmica existe a limitação de escalabilidade quando pretendemos executar sobre um grande número de aplicações.

Nesta forma de análise existem também limitações à execução de aplicações em ambientes virtuais ou privilegiados Babil et al. (2013) e Sarwar et al. (2013). O que levou a novas técnicas de análise recorrendo ao uso de Virtual Private Network (VPN) para suportar o estudo das comunicações de rede, conforme descrito por Le et al. (2015), Rao et al. (2012) e Razaghpanah, Vallina-Rodriguez et al. (2015).

2 Pidcat Github - Disponível em: https://github.com/JakeWharton/pidcat - Acedido a 20 Out. 2022

Outras limitações deste tipo de análise prendem-se sobretudo com técnicas contra ataques de Man-In-The-Middle (MITM) Fahl et al. (2012), Georgiev et al. (2012) e Razaghpanah, Niaki et al. (2017) e a já referida dificuldade de escalar para múltiplas aplicações ou com grande complexidade, que resulta da necessidade de existir interação do utilizador com a aplicação para *input* de dados.

2.4.3 *Análise de tráfego de rede*

É importante para o analista que pretenda perceber todo funcionamento de uma aplicação não só compreender o fluxo de dados, respeitante ao funcionamento do código da aplicação, mas também estudar o tráfego de rede que é resultado da sua execução.

Para analisar o tráfego de rede existem duas abordagens que iram ser discutidas:

1. Análise do tráfego em tempo real, através de um ataque MITM;
2. *Packet Dump* do tráfego de rede, para posterior análise.

Em ambas as opções se o tráfego de rede não estiver encriptado é fácil de analisar o seu conteúdo. Vai ser demonstrado o processo utilizado para as duas abordagens referidas.

O objetivo é perceber se existem interações com *APIs* externas, e se é possível intercetar as mesmas de forma a analisar o seu conteúdo e finalidade. Desta forma podemos identificar se existe comunicação com domínio(s) que não seja(m) confiável(is), bem como perceber se existem comunicações que não estão protegidas de forma correta. Dados presentes do DM, pertencentes ao seu utilizador são especialmente importantes se estivermos a falar de dados privados, dados de autenticação ou transferência de componentes dinâmicos que alterem o comportamento da aplicação.

Com este capítulo foi possível dar a conhecer ao leitor as principais diferenças das análises a dispositivo móveis, referindo as vantagens de desvantagens para cada uma destas. Foi também feito um resumo, destacando alguns trabalhos na área forense que analisam aplicações de *ride sharing*.

Tabela 2: Resumo das dificuldades em investigações forenses móveis.

Fonte: (Tamma et al., 2020)

Dificuldade	**Descrição**
Diferenças de Hardware	Existem diferenças entre modelos e fabricantes de DM. Isto origina a necessidade de manter a par das novas tecnologias e procedimentos necessários para os diferentes dispositivos.
SO's Móveis	Além de existirem diversos SO, entre os quais *iOS, Android, BlackBerry OS, Windows Phone OS, webOS*. Cada um destes tem associado diferentes versões com particularidades na sua implementação, tornando assim a tarefa ainda mais difícil.
Definições de Segurança	Proteções existentes à segurança dos dados e privacidade do seu utilizador através da sua encriptação. A necessidade de quebra dos mecanismos de encriptação em uso pode ser necessário para a obtenção de prova.
Prevenção alteração	A não alteração da prova é um conceito base, no entanto ao ligar o dispositivo pode levar à sua modificação. Mesmo com o dispositivo desligado em muitos casos a aplicação associada ao alarme ainda continua a funcionar, o que pode resultar na perda ou modificação de dados.
Técnicas anti-forense	Tais como omissão de dados, obfuscação de dados, alteração de dados, remoção segura de dados torna a investigação mais complexa.
Código desbloqueio	Se o dispositivo estiver protegido por um código de bloqueio pode ser necessário obter acesso sem alterar os dados do dispositivo. Poderá nem sempre ser possível, dependendo da versão do SO atingir esse fim sem alteração dos dados.
Falta de recursos	Como referido anteriormente devido à diversidade encontrada, é necessário um grande conjunto de acessórios forenses para auxiliar o processo de aquisição.
Natureza dinâmica da evidência	Interagir com uma aplicação pode alterar inadvertidamente os dados guardados pela mesma.
Reset acidental	Tal como o nome indica, muitos destes dispositivos dispõem de opção para apagar/*reset* ao seu conteúdo.
Proteger comunicações	Como dispositivos móveis podem ser utilizados para comunicar com diferentes tecnologias (Wifi, Bluetooth, infra-vermelhos, redes móveis) é importante eliminar todas as possibilidades de comunicação após a sua apreensão
Falta de ferramentas	A inexistência de uma ferramenta de análise forense digital comum a todos os dispositivos, obriga à identificação e escolha da ferramenta de acordo com o dispositivo alvo.
Programas maliciosos Aspetos Legais	Dispositivos móveis podem estar envolvidos em crimes cometidos por diferentes fronteiras geográficas. O que obriga ao analista forense conhecer a natureza do crime e as leis locais.

Tabela 3: Detalhes das aquisições efetuadas

Fonte: (Matulis e Karabiyik, 2022)

Aquisição	**Software Forense**	**Objetivo**
Lógica #1	`Magnet AXIOM`	Obter dados pré-existentes
Cloud #1	`Magnet AXIOM`	Obter dados pré-existentes
Lógica #2	`Cellebrite UFED`	Obter dados decorrentes do uso da aplicação
Cloud #2	`Magnet AXIOM`	Obter dados decorrentes do uso da aplicação
Física #3	`Magnet AXIOM`	Obter uma imagem bit a bit após utilização

3

CONCEITOS E TECNOLOGIAS

Este capítulo explora um panorama abrangente de conceitos e tecnologias fundamentais para uma análise detalhada de uma aplicação móvel no ecossistema *Android*. Esta jornada começa por uma exploração minuciosa dos aspectos fundamentais da plataforma *Android*, compreendendo o sistema de ficheiros e o seu ecossistema. Por fim são apresentados aspetos de segurança os quais incluem a arquitetura, aplicações e rede em *Android*. Este capítulo servirá como base sólida para a análise posterior, fornecendo conceitos cruciais sobre como interagir e avaliar o funcionamento interno de aplicações móveis no contexto *Android*.

3.1 ANDROID FILE SYSTEM

O SO *Android* foi criado com base no Linux, utilizando a mesma hierarquia de ficheiros, com as devidas alterações para se adequar à estrutura pretendida por parte do fabricante de dispositivos móveis. Sendo que no topo da árvore da hierarquia de ficheiros se situa a denominada */ root*. Existem sete partições principais comuns a todos os dispositivos *Android*, incluindo a partição para cartão Secure Digital Card (SD). Estas encontram-se localizadas abaixo da partição de *root*:

- `/boot:` Para os ficheiros e a informação necessária após ligar o dispositivo móvel, iniciar o processo de `boot`. Este contém o *Kernel* e Random Access Memory (RAM), ambos com informação de valor forense que pode ser capturada.
- `/system:` Contém ficheiros do SO em Read Only Memory (ROM) além dos presentes do `/boot`. Estes ficheiros contém a interface do utilizador, bem como as aplicações pré-instaladas no dispositivo.
- `/recovery:` Ferramentas aqui presentes ajudam o utilizador a guardar cópias para restauração de outras partições. Aqui encontram-se também os ficheiros de `boot` no caso de iniciar o mesmo no modo de recuperação.
- `/data:` Uma das mais importantes partições para se compreender o funcionamento de uma aplicação. Aqui são guardados os dados utilizados pelas aplicações pré-instaladas como contactos, histórico de chamadas, Short Mes-

sage Service (SMS), etc., bem como as instaladas pelo utilizador. O acesso a esta partição pode ser feita através de uma ligação a um computador, em que o nível de acesso é o mesmo do utilizador. Se for feito o *factory reset*, todos os dados do utilizador são eliminados desta partição.

- `/cache`: Tal como o nome indica, esta pasta guarda os dados das aplicações que são mais frequentemente utilizadas, o que permite um aumento do desempenho do dispositivo. Os conteúdos desta partição podem ser eliminados, dado que os mesmos são reconstruidos durante o normal uso do dispositivo.
- `/misc`: Contém definições do sistema e *hardware*, o que se pode traduzir no estado ligado/desligado. Se esta partição não existir ou estiver corrupta, diversas funcionalidades do mesmo poderá não funcionar, ou funcionar de forma incorreta.
- `/sdcard`: Originalmente esta nomenclatura era usada para se referir aos dados presentes no cartão SD físico. No entanto ao longo do tempo esta passou a ser utilizada como armazenamento interno emulado, ou seja mesmo na ausência de um cartão SD físico. Esta pode guardar diversos formatos de dados, desde imagens, fotos captadas pela câmara, ficheiros do utilizador ou aplicações, etc..

O sistema de ficheiros *Android* consegue suportar diversas técnicas de armazenamento de dados, organização e obtenção de informação do suporte de dados. O *Android* pode suportar diversos ficheiros de sistemas, o qual pode suportar diversas partições, onde cada uma destas pode suportar diferentes sistemas de ficheiros. O sistema de ficheiros *Android* é organizado em três categorias principais, sistemas de ficheiros para memória *flash*, sistemas de ficheiros de disco (*media-based*) e sistemas de pseudo ficheiros (Umer Farooq, 2018).

3.2 ECOSSISTEMA ANDROID

O ecossistema *Android* (figura 1) é composto por três camadas, o qual inclui: uma camada de baixo nível que contém o sistema operativo e a interface que permite aceder aos recursos do dispositivo. Uma camada intermédia que consiste nas bibliotecas e na *framework* de aplicação que contém grande parte das funcionalidades da API. E por último, na camada mais alta, designada por camada de aplicação, residem todas as aplicações instaladas pelo utilizador e sistema. Esta última camada

é responsável por permitir ao utilizador interagir com o dispositivo móvel (Chantzis et al., 2020).

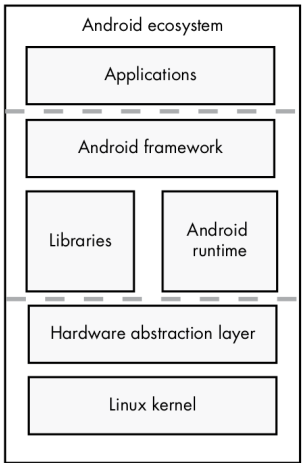


Figura 1: Ecossistema Android

Fonte: (Chantzis et al., 2020)

Este ecossistema permite flexibilidade tanto aos seus utilizadores, como programadores,possibilitando a instalação de aplicações de origem não confiável, em que se destacam o uso de extensões ou jogos. Os possíveis adversários podem desta forma ludibriar os seus utilizadores a instalar *malware*, camuflado em aplicações confiáveis, o qual pode fornecer acesso a recursos sobre os quais não deveria ter acesso ou permissão. Estas vulnerabilidades podem ser expostas devido à falta de conhecimento ou descuido por parte dos programadores, deixando assim expostos dados sensíveis devido à falta de controlos de segurança efetivos, ou desativação dos mesmos. A existência de vulnerabilidades é principalmente preocupante quando falamos de dispositivos mais antigos, com versões completamente desatualizadas, o que com dispositivos móveis ocorre em poucos anos. Este tipo de problema é conhecido como *software fragmentation* (Chantzis et al., 2020).

3.3 ARQUITETURA DE SISTEMA DE ANDROID

Tal como é possível verificar na figura 2, o SO *Android* criado pela Google para dispositivos móveis é construído sobre o *Kernel* Linux, o qual é uma plataforma *open source*. Podemos verificar que a arquitetura *Android* é composta por cinco componentes principais, os quais são descritos de seguida:

- Aplicações de Sistema
- Estrutura de Aplicações

- *Android Runtime*
- Bibliotecas Nativas
- *Kernel* Linux

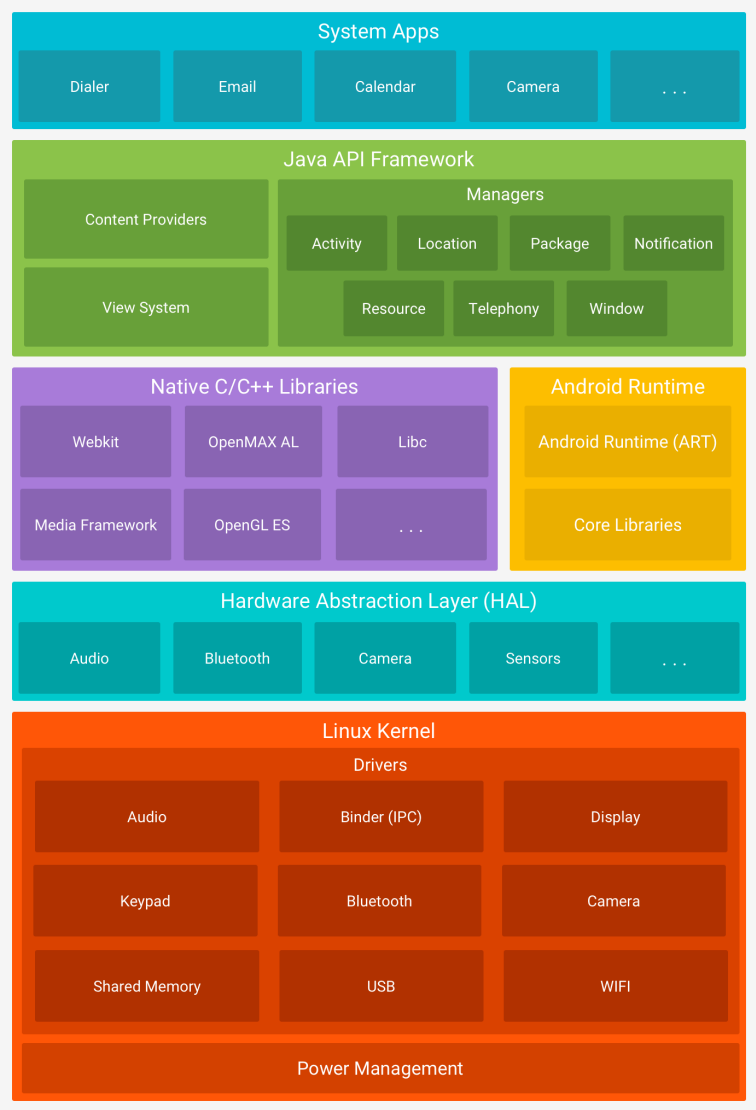


Figura 2: Arquitetura Android
Fonte: (Moreb, 2022)

3.3.1 *Kernel Linux*

Tal como outros sistemas Unix, o *Kernel* dispõem dos seus próprios *drivers* para diversas tarefas, tais como comunicar com *hardware*, acesso à rede, acesso a ficheiros de sistema e gestão de processos. Contudo o *Kernel* existente em *Android* apresenta as suas próprias particularidades inerentes às características necessárias para dispositivos móveis, em contrapartida à versão para *desktop* ou servidor.

Nestas característica distintas também designadas por *Androidisms* Yaghmour (2012), podemos citar os mais relevantes:

- **Low Memory Killer Daemon (LMKD):** Este componente realiza uma análise constante análise do uso em memória de todos os processos em execução. Identifica o(s) processo(s) que está a fazer uso de grandes quantidades de memória e termina (*kill*) o que menor impacto tem no sistema, de forma a manter o sistema com níveis aceitáveis de desempenho.
- ***Wakelocks*:** Este é um mecanismo de gestão de consumo de energia. Ele pode forçar o dispositivo a manter-se "ativo", e não entrar em modo de baixo consumo (mantém a aplicação ativa). É definido através da permissão no manifest *«uses-permission> android.permission.WAKE_LOCK"*[1].
- **Alarmes:** Os alarmes são utilizados para agendar tarefas a serem executadas em momentos específicos e em intervalos regulares. São também utilizados para ativar a execução de código em segundo plano, bem como responsáveis pela notificação das aplicações de acordo com os eventos definidos.
- ***Paranod Networking:*** Esta característica limita as operações de rede apenas a grupos específicos (*group IDs*).
- **Memória anónima partilhada (*ashmem*):** Este é um mecanismo de gestão de memória que permite a partilha de memória entre processos. O seu uso em *Android* permite a otimização da comunicação entre processos através do *Binder*.
- ***Binder***: Mecanismo para comunicação entre processos, que garante a sua segurança e validação.

1 "WakeLock", documentação - Disponível em: https://developer.android.com/reference/kotlin/android/os/PowerManager.WakeLock - Acedido a 12 de Out. 2022

3.3.2 *Binder*

De uma forma resumida, o *Binder* é um *driver* ao nível do **Kernel**, responsável por mover dados (*data*), em memória entre processos.

Para ser perceber o funcionamento do *Binder* é preciso perceber que este é um mecanismo que utiliza Inter-process communication (IPC). O IPC é a ponte que permite os processos conhecerem-se uns aos outros. Em *Android* tal como outros sistemas Unix, é necessário garantir que um processo não acede diretamente à memória de outro processo, tanto por razões de segurança como de estabilidade.

Quando um processo envia uma mensagem a outro, o *Kernel* é o responsável pela sua entrega. O *Binder* por sua vez obtém a lista de permissões de quem faz o pedido (*caller*). Se a permissão existir o *kernel* aloca o espaço necessário na memória do processo destino. Copiando a mensagem para a memória do processo destino, e notificando o mesmo da sua recepção. O *Binder* garante que os *ID's* utilizados não são falsificados, para que as aplicações possam agir apenas de acordo com as suas permissões (Elenkov, 2015).

Por exemplo, apenas processos com privilégios específicos de sistema podem realizar ações como ativar o *Bluetooth* sem requerer a intervenção do seu utilizador. Esta ação apenas pode ser efetuado por um processo com *system ID*.

Em resumo, o *Binder* e os mecanismos de IPC desempenham um papel crucial ao permitir a comunicação e a colaboração entre processos em SO *Android*. Este gere desde permissões, alocação de memória e transferência de mensagens entre diferentes processos. O *Binder* é também responsável por garantir que essa comunicação seja segura e controlada mantendo o isolamento necessário entre processos.

3.3.3 *Hardware Abstraction Layer - HAL*

Esta camada é responsável por permitir a comunicação independentemente do hardware (agnóstico) e o sistema operativo *Android*, através do uso de diferentes bibliotecas. Deste modo não existe a necessidade de alterar o funcionamento entre a camada de alto nível e as implementações de baixo nível dos drivers (Moreb, 2022).

3.3.4 *Native Userspace*

Por cima do *Kernel* existe a camada destinada ao espaço do utilizador nativo, esta é composta por bibliotecas, *daemons* nativos e o binário *init*. Este último é o primeiro processo a ser iniciado, responsável por iniciar todos os restantes processos. Esta camada é responsável por correr processos do "utilizador"no seu próprio espaço de memória virtual fora do espaço alocado para o *Kernel* do sistema operativo.

3.3.5 *Dalvik Virtual Machine (DVM) / Android RunTime (ART)*

Tal como referido anteriormente grande parte de *Android* é desenvolvido em Java, que por sua vez é executado dentro de uma máquina virtual em Java (Java Virtual Machine (JVM)). Ao compilar um programa em Java obtemos **bytecode**, o qual pode ser executado por uma JVM. No entanto foi criado para *Android* uma máquina virtual designada por Dalvik Virtual Machine (DVM), até ao *Android* 5.0, para executar as aplicações instaladas no dispositivo. Cada aplicação *Android* corre na sua própria instância de DVM, este é um aspeto crucial de segurança em *Android*. Android Runtime (ART) e o seu predecessor DVM foram criados especificamente para serem usados por dispositivos *Android*. Ambos são compatíveis e podem executar *Dex bytecode*, como tal aplicações desenvolvidas para Dalvik podem funcionar com ART. Podemos observar as diferenças na figura 3 do funcionamento entre uma máquina virtual Java e uma máquina virtual *Dalvik*.

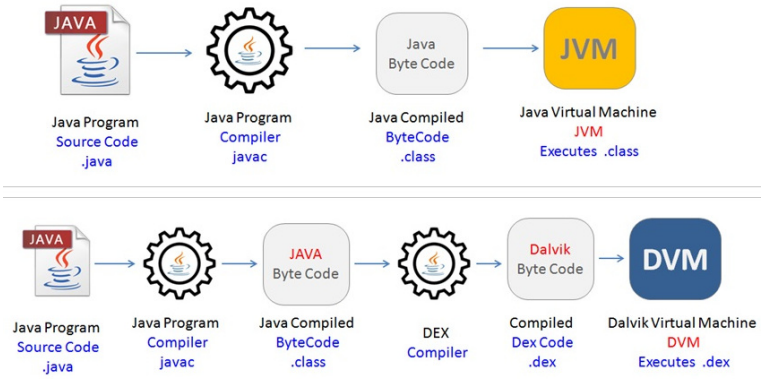


Figura 3: Diferenças de compilação JVM e DVM

Fonte: www.learncomputerscienceonline.com/android-development/

Dalvik

DVM corre *Dalvik Bytecode*, o qual é Java *bytecode* convertido pelo compilador

Dalvik Executable (DEX). Este compilador converte ficheiros *.class* em ficheiros *dex* utilizando a ferramenta *dx* (Tamma et al., 2018). *Dalvik* foi criado para resolver o problema que era a falta de memória RAM, nos primeiros dispositivos móveis otimizando o seu uso. Em vez de compilar toda a aplicação para código máquina antes da sua execução, usava a técnica de compilação Just in Time (JIT). Neste modo o compilador funciona com um interpretador, o qual interpreta pequenos pedaços de código durante a execução da aplicação, conhecido como compilação em execução (*runtime*). Esta solução trouxe alguns problemas, tais como o aumento do tempo de execução, o qual sofreu alterações para melhorar o seu desempenho, como colocar em *cache* as instruções usadas mais frequentemente. Com o aumento do tamanho das aplicações e a sua necessidade de correr mais instruções em *runtime*, aliado ao fraco desempenho que era demonstrado e com a limitação da RAM a deixar de ser um problema em novos dispositivos móveis, Dalvik caiu em desuso com a criação do ART.

ART - Android Runtime

ART é um paradigma oposto ao apresentado pelo *Dalvik*, disponibilizado a partir do *Android* 5.0 *Lollipop*. Enquanto o *Dalvik* utiliza um compilador JIT, que compila o código em tempo de execução, o ART adota um compilador Ahead of Time (AOT), que compila o código antes da execução da aplicação, mantendo as instruções em memória. Essa abordagem melhorou significativamente o desempenho em tempo de execução em comparação com seu antecessor JIT. No entanto também existem algumas desvantagens em relação ao seu antecessor. Uma das desvantagens é o maior uso de memória, uma vez que o código é pré-convertido para instruções nativas, que podem ocupar mais espaço do que o *bytecode* original. Além disso o tempo de instalação da aplicação é superior uma vez que toda aplicação tem que ser transformada para código máquina, durante a sua instalação (antes da primeira execução).

O ART é composto por um compilador denominado por **dex2oat**, o qual aceita ficheiros DEX como *input* e gera um executável compilado da aplicação otimizado para o dispositivo. Com o ART, os ficheiros otimizados *dex* (.odex) são substituídos por executáveis no formato Executable and linking format (ELF) (Tamma et al., 2018).

Profile Guided Compilation

Nem sempre a compilação AOT é vantajosa, como tal o conceito de JIT foi reintro-

duzido, com modificações adicionadas e um novo termo Profile Guided Optimization (PGO)[2].

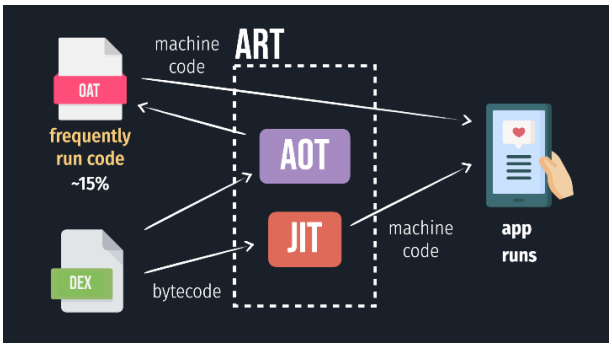


Figura 4: Compilação Profile Guided

Fonte: https://proandroiddev.com/android-runtime-how-dalvik-and-art-work-6e57cf1c50e5

Foi na sua versão 7 do Android (*Nougat*), em 2016 que ditou o regresso do JIT, ao qual adicionou o PGO. Esta é uma estratégia que permite otimizar constantemente o desempenho da aplicação após cada utilização, focando na otimização das funcionalidades frequentemente utilizadas.

O seu funcionamento pode ser observado em *cache*, onde os caminhos (*path*) mais utilizados são guardados e otimizados recorrendo a uma ferramenta designada por *dex2opt*.

A pré-compilação das partes mais utilizadas de uma aplicação é apenas efetuada quando o dispositivo está inativo e em carregamento, para reduzir o consumo energético. Queremos com isto dizer que o compilador otimiza seletivamente partes do código que são frequentemente utilizadas, deixando de fora partes que são raramente ou nunca utilizadas. Reduzindo assim o tamanho do executável e economizando recursos de compilação. Com esta solução híbrida e dinâmica podemos ter o melhor de dois mundos de forma a ter o melhor desempenho possível, fazendo uso assim da melhor forma da RAM disponível e dos recursos do dispositivo. A única limitação neste uso seria quando o utilizador faz uso da aplicação pela primeira vez, como não existe conhecimento dos métodos mais utilizados o compilador utilizado será o JIT. Para resolver esta situação a Google desenvolveu o *Profiles in the cloud*, este foi introduzido em 2018 na versão 9 do *Android*.

Os *Profiles in the cloud* funcionam através da recolha e envio para a *Play Store* de perfis de uma aplicação e dispositivo específico. Onde eles são agregados e posteriormente distribuídos e aplicados durante a sua instalação.

2 How Dalvik and ART work - Disponível em: https://proandroiddev.com/android-runtime-how-dalvik-and-art-work-6e57cf1c50e5 - Acedido a 13 de Out. 2022

Após a instalação é recolhida informação do utilizador e dispositivo ligada à aplicação, num ficheiro designado por *Core Profile*. E utilizada quando um novo utilizador instalar a aplicação pela primeira vez este ficheiro é descarregado em conjunto com o Android Application Package (APK) (APK + *Core Profile*). O ART irá utilizar essa informação para compilar AOT os métodos mais utilizados por outros utilizadores. Assim o utilizador pode ter de forma imediata o melhor desempenho desde a primeira utilização da aplicação. Vai ser então repetido o processo, através de nova recolhida de informação sobre o uso do novo utilizador da aplicação, incluindo em *idle* e partilhada com outros novos utilizadores da mesma.

3.3.6 *Dalvik e Smali*

É importante perceber quando estamos a tentar reverter o processo através de técnicas de engenharia reversa, estamos a converter o *DEX bytecode*, para *Smali* e por último obtemos o código fonte em Java após a descompilação do *Smali*.

Smali é a versão legível para humanos de *Dalvik bytecode*, o termo *Smali* é usado em dois sentidos, para se referir tanto ao código produzido, como à ferramenta. *Smali* e *Baksmali* são também estes, os nomes das ferramentas utilizadas como *assembler e disassembler* respetivamente. *Smali* é comparado à linguagem *assembly*, uma vez que tal como este é usado entre o código de alto nível e o *bytecode*.

3.3.7 *Bibliotecas nativas C/C++*

O sistema operativo *Android* inclui um conjunto de bibliotecas nativas em C e C++ *(Media, Graphics, Surface Manager, OpenGL, etc.)* para serem utilizadas por diferentes componentes e serviços (exemplo: Hardware Abstraction Layer (HAL) e ART), também estes provenientes de código nativo em C e C++ (Skulkin et al., 2018). Este código está otimizado para o *hardware*, ao contrário de aplicações para *Android* e sua *framework*, a qual é escrita em Java. Algumas destas bibliotecas podem ser acedidas através de Application Programming Interface (API)'s existentes na *framework* Java fornecida pela plataforma *Android*. É também disponibilizado para programadores através do *Android* Native Development Kit (NDK)[3], a possibilidade de referenciar código nativo em C ou C++, fazendo uso das bibliotecas nativas existentes.

3 Android NDK, Documentação - https://developer.android.com/ndk/guides - Acedido a 18 de Outubro de 2022

3.3.8 *Java API Framework*

Esta camada é responsável por fornecer API em Java que garantem os serviços e funções básicas do SO *Android* e do dispositivo. Programadores podem então utilizar estas API para acederem às funções básicas essenciais para construir aplicações *Android*, tais como as classes para *activities, services e content providers*, disponíveis nos pacotes *android.app.**; Ferramentas de Graphical User Interface (GUI), (*android.view.* e android.widget*) e classes para acesso a ficheiros e bases de dados (*android.database e android.content.**). Também inclui classes que permitem interagir com o *hardware* do dispositivo, incluindo classes que permitem tirar partido de serviços de alto nível oferecidos pelo sistema (Elenkov, 2015).

A Tabela 4 (página 31), apresenta os serviços fornecidos pela *Android Application Framework*. Esses serviços compreendem um conjunto essencial de classes, com ênfase nos seguintes componentes-chave: *Activity Manager*, *Resource Manager*, *Location Manager* e *Notification Manager*.

Tabela 4: Serviços prestados através da Android Application Framework

Fonte: (Anmol Misra, 2016)

Serviço	**Descrição**
Activity Manager	Gere o ciclo de vida das atividade nas aplicações e outros componentes da aplicação. Quando uma aplicação pede para começar uma atividade, através de *startActivity()* é o *Activity Manager* que garante este serviço.
Resource Manager	Garante o acesso a recursos os quais podem ser *strings*, gráficos, ícones, áudio ou ficheiros de *layout* que definem a estrutura gráfica da aplicação. Podemos dizer que este serviço garante que os recursos corretos são entregues aos utilizadores.
Location Manager	Grande parte dos dispositivos *Android* vem equipado com Global Positioning System (GPS), utilizando a localização através de informação por satélite que permite precisão de alguns metros. As aplicações podem assim requerer permissão ao seu utilizador para obter essa informação. Este serviço pode também fazer uso da tecnologia wireless se o seu utilizador estiver num espaço fechado para obter um melhor posicionamento do dispositivo.
Notification Manager	Aplicações que necessitem de obter notificações sobre eventos específicos, recebem essa informação através deste serviço.
Package Manager	*Android* mantém informação sobre as aplicações instaladas, desde instalação e/ou desinstalação, requisição de permissões, consumo de memória. Esta informação pode ser usada pelos programadores de forma a identificar funcionalidades que podem ou não ser ativadas/desativadas.
Content Providers	Permite partilhar dados com outras aplicações ou aceder a dados de outras aplicações.
View System	*Android* providencia formas de criar facilmente componentes visuais, necessários para a interação com aplicação. Estes componentes incluem as mais diversas ferramentas de forma a criar o seu próprio ambiente gráfico que seja adequado à aplicação, de acordo com os objetivos pretendidos.

3.3.9 *Aplicações de Sistema*

Esta camada é responsável por permitir a interação direta entre utilizador e dispositivo. Contém um conjunto de aplicações consideradas essenciais para o seu utilizador, como o calendário, *browser*, aplicação de email, SMS, entre outras. O SO *Android* não diferencia nem altera o comportamento entre aplicações que são instaladas pelo utilizador, através do Google Play e aplicações que vêm instaladas por omissão com o dispositivo. Esta camada é responsável pela gestão da aplicação por omissão que é usada, seja esta instalada pelo utilizador ou "nativa". A interação do utilizador com algumas das aplicações instaladas envolve o uso direto da camada de aplicações do sistema, tais como escrever ou enviar mensagens.

3.4 SEGURANÇA DO SISTEMA OPERATIVO ANDROID

A criação de qualquer arquitetura de segurança de SO é desenvolvida para assegurar a segurança do próprio dispositivo. Em sistemas tradicionais aqui descritos como fixos, essa segurança está implícita nas suas características e localização do próprio dispositivo, o qual se encontra restringido a uma divisão fechada e inamovível. Devido às suas próprias características físicas, os dispositivos móveis são suscetíveis de serem roubados ou de fácil manipulação por terceiros, sendo estes também facilmente danificados ou perdidos. Sabendo destas limitações, existem diversas características de segurança implementadas, e em constante evolução que permitem a segurança dos dados presentes no dispositivo móvel.

3.4.1 *Verified Boot*

O *verified boot* garante de forma criptográfica que o código do sistema e os dados provêm de uma fonte confiável (Original Equipment Manufacturer (OEM)). O *verified boot* estabelece um cadeia de confiança, que começa desde a verificação do hardware até ao *bootloader*, que se estende à partição de *boot* e às restantes partições. Durante a fase de *boot* do dispositivo, cada fase verifica a integridade e autenticidade da próxima fase antes de iniciar a sua execução.[4];

4 Verified Boot, documentação - Disponível em: https://source.android.com/docs/security/features/verifiedboot - Acedido a 22 Out. 2022

3.4.2 *Trusted Execution Environment*

O Trusted Execution Environment (TEE) é um conceito genérico que se refere a um ambiente seguro de execução em um sistema computacional. Esse ambiente está isolado do sistema operativo principal e é projetado para executar código de maneira segura e confiável, protegendo informações e operações críticas. O TEE é uma abstração mais ampla que pode ser implementada de várias maneiras em diferentes sistemas e dispositivos. Se estivermos a falar de um dispositivo *Android* versão 8 ou superior, o TEE é obrigatório para se poder utilizar as aplicações e serviços da Google considerados como essenciais Google Mobile Services (GMS). O TEE para os dispositivos que implementam o mesmo não é um componente físico à parte do dispositivo, podendo apresentar-se sobre a forma de *hardware* ou *firmware*. A localização exata do mesmo pode variar dependendo do modelo e fabricante, no entanto na maioria dos dispositivos está integrado no Central Processing Unit (CPU).

A Google criou um SO que oferece um ambiente de execução confiável (TEE) designado por *Trusty*, específico para *Android*. O *trusty* TEE foi criado para ser usado como uma norma, eliminando a o uso de outras implementações de terceiros e assim evitar o risco de incompatibilidade e a introdução de vulnerabilidades de segurança. O *Trusty* é composto por três componentes, o primeiro derivado do *Little Kernel*[5], em segundo um driver do *kernel Linux* que permite transferir dados ente o TEE e *Android*, por último uma biblioteca que permita comunicar com aplicações confiáveis através do *kernel Linux*[6].

Numa implementação TEE, o processador principal é considerado como "não confiável", não podendo aceder a certas áreas da RAM, registos de *hardware*, segredos específicos do dispositivo, como chaves criptográficas guardadas pelo fabricante.

A solução encontrada foi virtualizar o processador principal, criando assim um segundo ambiente virtual isolado do processador principal. Este segundo ambiente é considerado seguro, o que permite a separação de código que seja considerado como não seguro. Para implementar esta funcionalidade é empregue o *secure hardware*. Cabe ao software a ser executado no processador principal delegar operações que necessitem do uso de dados sensíveis/secretos ao TEE.

5 Little Kernel- Disponível em: https://github.com/littlekernel/lk - Acedido a 15 Mar. 2024
6 Trusty Android AOSP Docs- Disponível em:https://source.android.com/docs/security/features/trusty?hl=pt - Acedido a 14 Mar. 2024

3.4.3 *Android Keystore System*

O *Android keystore system* representa uma das formas fundamentais de proteção de dados nos dispositivos.

O uso deste mecanismo está ligado ao armazenamento seguro de chaves criptográficas, proteção de dados sensíveis e realização de operações criptográficas através de um ambiente seguro.

Para garantir o isolamento durante as funções de criptografia é utilizado o TEE, descrito anteriormente para garantir o isolamento das suas operações criptográficas. Para se fazer uso da mesma deve ser utilizada a API correspondente, a qual permite aceder às chaves criptográficas nesta contida. O acesso a estas chaves pode ser feita através de diferentes formas de autenticação (biometria, PIN, padrão de desbloqueio), com o objetivo de assegurar a proteção das chaves mestres. As chaves mestres são utilizadas para garantir a segurança das restantes chaves armazenadas no *keystore*.

O sistema *keystore* não permite que seja autorizada a extração de chaves neste contidas. Podemos assim fazer uso das mesmas, mas não é permitida a sua exportação e/ou uso fora do DM *Android*. O *Android keystore* obriga também as aplicações que fazem uso das chaves nele contidas a especificar a permissão de uso, restringindo o seu acesso a processos de outras aplicações. Queremos com isto dizer que as aplicações devem definir restrições específicas sobre a manipulação e uso das chaves armazenadas *Android Keystore*. Esta abordagem visa garantir a segurança e integridade das chaves, limitando o seu acesso a processos que tenham como origem aplicação que o despoletou.

O *Android keystore* utiliza duas formas de proteção contra a extração das suas chaves:

- As chaves a serem utilizadas nunca entram no processo da aplicação. Para que seja garantido a sua segurança um processo em segundo plano é que realiza as operações criptográficas. Isto garante que mesmo o atacante com acesso ao processo da aplicação, apenas pode ter acesso às chaves utilizadas pela aplicação, mas não poderá extrair o seu conteúdo.
- O material utilizado pela chave deve estar vinculado a um hardware seguro TEE, ou a um elemento de segurança Secure Element (SE), garantindo o isolamento durante a sua execução. Ao utilizar esta segunda forma, os componentes da chave nunca são expostos fora do *hardware* seguro.

A *keystore* foi introduzida desde a API 1 para *Android*, enquanto o *Android Keystore Provider* está disponível desde a API 18. Mais tarde, no *Android 9* (API 28) a Google introduziu o *StrongBox*, este último para dispositivos com *secure chip*. Apesar do TEE ser bastante seguro, o *StrongBox* é ainda mais seguro, dispondo do seu próprio *secure chip* com um CPU e memória própria.

3.4.4 *Tamper-resistant Hardware support*

O *tamper-resistant hardware support* foi adicionado ao *Android* 8.0 para uma maior segurança, o qual é utilizado para verificar a *passcode* após a sua introdução no *lock screen*. Se a verificação for bem sucedida é devolvido um segredo de alta entropia que pode ser usado para derivar a chave de encriptação para ter acesso ao conteúdo em disco.

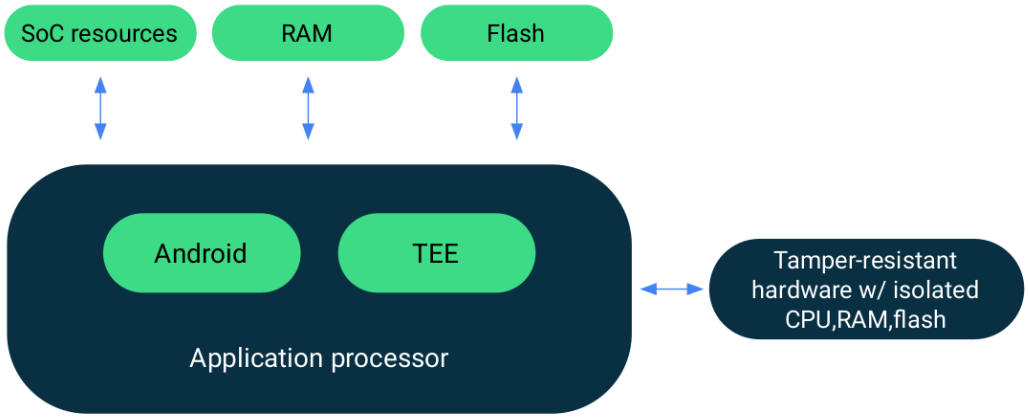


Figura 5: Proteção por hardware auxilia em diferentes proteções para o dispositivo

Fonte: https://source.android.com/docs/security/overview/reports

3.4.5 *Protected confirmation*

Android Protected Confirmation é uma característica de segurança que foi introduzida a partir da versão do *Android* 9.0, o qual utiliza em conjunto uma funcionalidade designada por *Trusted UI* para confirmar uma ação por parte do seu utilizador.

A sua principal função é impedir que aplicações fraudulentas ou SO comprometidos possam realizar operações críticas inadvertidamente.

Quando uma aplicação invoca o uso da *Protected Confirmation* o controlo é passado à *Trusted UI* onde os dados da transação são identificados e posteriormente conferidos pelo seu utilizador.

O seu uso é encontrado em transferências bancárias entre utilizadores, autenticação de utilizador e outras situações em que seja necessário a confirmação do utilizador.

3.5 SEGURANÇA DO SISTEMA OPERATIVO

O modelo de segurança *Android*, confia no *kernel Linux*. Tal como encontramos em outros sistemas *Unix*, o *kernel* isola os dados e processos. No entanto ao contrário de sistemas computacionais tradicionais, em *Android* é feita a distinção entre aplicações ao invés da distinção entre utilizadores.

3.5.1 *Utilização de UID*

Existe o conceito de separação de User Identification (UID), em que cada utilizador tem o seu próprio identificador único. Esta associação toma efeito durante a instalação da aplicação, desta forma apenas pode interagir com ficheiros que sejam proprietários do utilizador. Estas medidas é para evitar que outro utilizador tenha acesso indevido a informação manipulada/criada pela aplicação.

No entanto é possível partilhar o UID com outra aplicação, apesar deste tipo de comportamento não ser encorajado, uma vez que se existir comprometimento dos dados de uma aplicação, ambas ficam comprometidas. Esta situação encontra-se descrita no *Android* `manifest`, através da atribuição *android:sharedUserId*.

3.5.2 *Sandbox*

O *Android* corre num sistema de múltiplos utilizadores em Linux, o que significa que cada aplicação tem a sua *sandbox* de segurança dedicada, e por norma o seu funcionamento não afeta o comportamento de outras aplicações que pretendam partilhar os seus dados. Isto é conseguido ao executar em separado os processos de cada aplicação, com o seu respetivo ID do utilizador(UID) (Elenkov, 2015). Desde o nível 21 da API (*Android* 5), é utilizado o ART, ao invés do DVM, conforme anteriormente referido. O seu funcionamento é bastante parecido, em

que ambos correm uma aplicação através de código compilado em Dalvik. No entanto é considerado que ART tem um melhor desempenho em relação ao seu predecessor DVM. Desde o *Android* 5.0(Lollipop) é obrigatório respeitar a *Security-Enhanced Linux (SELinux)*, o qual nega a interação entre processos. Para que seja possível interagir entre os processos é necessário definir políticas que permitam apenas interações estritamente necessárias entre processos. Esta afirmação aplica-se inclusive a processos que estejam a correr com privilégios de *root/superuser*.

Estas VM's criadas para aplicações tem apenas acesso a componente que sejam necessário para a sua execução (política de menor privilégio "*least privilege*"). Estas VM's associadas a processos individuais são criadas pelo *Zygote*[7].

3.5.3 *Anti-Exploitation*

O *Android* utiliza um conjunto de mecanismos que ajudam a proteger contra exploração de bugs ou de vulnerabilidades existentes, nomeadamente o Address Space Layout Randomization (ASLR) e o Data execution prevention (DEP).

O ASLR é uma técnica usada pelo SO *Android* para aleatorizar as localizações em memória do código e dados, de forma a dificultar os atacantes de explorarem vulnerabilidades. O ASLR torna mais difícil ao atacante prever onde determinados pedaços de código ou dados estão localizados, e consequentemente, mais complexo a sua exploração com sucesso por uma vulnerabilidade.

O DEP é outro recurso de segurança que ajuda a impedir que código malicioso seja executado num dispositivo *Android*. O DEP marca certas áreas da memória como não-executáveis, o que significa que qualquer tentativa por parte do código malicioso de executar sequência de octetos dessas áreas como se se tratasse de instruções será bloqueada. O DEP ajuda a proteger contra ataques *buffer overflow*, que por sua vez podem permitir aos atacantes injetarem código malicioso em aplicações vulneráveis e assumir o controlo do dispositivo.

7 "The Zygote Process", Medium - Disponível em: https://medium.com/masters-on-mobile/the-zygote-process-a5d4fc3503db - Acedido a 11 Out. 2022

3.5.4 *User and Data Privacy*

Proteger a privacidade do utilizador é fundamental em *Android*. A partir da versão 9.0 do *Android*, existem várias limitações que podem ser impostas de forma a aumentar a privacidade, tanto do dispositivo como do utilizador.

Destaca-se o limite de acesso por parte de aplicações em segundo plano aos sensores do dispositivo, bem como a restrição de informações que podem ser obtidas através de *scans Wi-Fi*. Quanto às limitações impostas que restringem a identificação do dispositivo, é exemplo o uso de endereços Media access control address (MAC) aleatórios aquando a procura por uma nova rede para a qual não existe no dispositivo uma associação prévia. Esta alteração ocorreu desde o *Android* 9, onde é possível utilizar um endereço MAC aleatório sempre que se liga a uma rede Wi-Fi, o que se tornou uma definição por omissão desde o *Android* 10.

Desde o *Android* 10 são também impostas restrições quanto à obtenção de dados e identificadores de sistema, acesso a informações da câmara fotográfica/vídeo/rede, bem como várias alterações no modelo das permissões.

O International Mobile Equipment Identity (IMEI) e *serial number*, são dados que podem identificar inequivocamente um dispositivo. O *Android* não permite que quaisquer aplicações possam aceder diretamente sem permissões para tal. A permissão em causa é `"READ_PHONE_STATE"`, no entanto esta permissão não se restringe ao IMEI, podendo ser também usada para obter toda a informação sobre o(s) número(s) de telefone(s) existente(s) (lista de contactos), ligações de rede móvel, bem como ter acesso ao registo de chamadas efetuadas.

Se pensarmos no IMEI, como um identificador persistente (não pode ser alterado), pode ser especialmente relevante no contexto de identificar hábitos de consumo. Esta informação é recolhida em conjunto com diversas fontes de dados, incluindo de desempenho do dispositivo e enviada para empresas de marketing/publicidade, tais como a *Adjust, TapJoy, AppsFlyer*, entre outras. Esta última (*AppsFlyer*) é utilizada pela aplicação em estudo *Bolt* para recolha de dados decorrentes da sua utilização. O IMEI é especialmente relevante quando existe um conjunto alargado de aplicações associado a um único IMEI, e assim obter um perfil de utilizador.

3.5.5 *Location Control*

As aplicações utilizam em muitos casos dados referentes à posição do seu dispositivo, acedendo a esses dados através de API's de localização. Uma das grandes alterações introduzidas no *Android* 10, e seguintes foi a possibilidade dos utilizadores serem notificados quando uma dada aplicação está a solicitar acesso à localização do seu dispositivo. Esta notificação permite ao utilizador definir quando a aplicação pode aceder a essas API's, apresentando as seguintes opções:

1. Apenas ao utilizar a aplicação;
2. Nunca;
3. Pode aceder sempre.

3.5.6 *Localização de dados*

Os dados gerados ou armazenados em dispositivos móveis podem ser encontrados em diferentes áreas físicas e lógicas.

Em seguida são enumeradas as principais localizações do dispositivo móvel, onde os dados com relevância forense podem ser encontrados:

Dispositivo móvel:

- **Armazenamento Interno do Dispositivo**: Dados como contactos, mensagens de texto, histórico de chamadas, correio electrónico, dados gerados ou obtidos por aplicações podem ser guardados internamente no dispositivo.
- **Cartão SIM**: Informações relacionadas com contactos e mensagens de texto podem ser mantidas no cartão SIM do dispositivo.
- **Armazenamento Externo (Cartão de Memória)**: Imagens, vídeos e documentos podem ser armazenados em cartão de memória externo, se o dispositivo suportar esse recurso.

As localizações anteriormente referidas são cruciais durante a fase de aquisição de evidências numa análise forense digital, uma vez que diferentes tipos de dados podem estar distribuídos em várias localizações em simultâneo.

No entanto existem outras localizações com interesse forense que podemos destacar, com a prevalência do armazenamento na *cloud*, é importante considerar também essa localização.

Outras localizações com interesse forense:

- **Armazenamento em Nuvem**: Com o uso de serviços de armazenamento em nuvem, como *Google Drive* para *Android* ou *ICloud* para *iPhone*, os dados podem ser sincronizados e armazenados remotamente.
- **Armazenamento em plataformas de rede sociais**: Dados relacionados a redes sociais, como mensagens, fotos e vídeos, podem estar disponíveis nas contas do utilizador nessas plataformas.
- **Correio electrónico**: Correio electrónico e seus anexos podem ser acedidos por meio de aplicações de *e-mail* e podem ser armazenados localmente e/ou na nuvem.

Ao realizar uma análise forense digital, devem ser examinadas todas as áreas de armazenamento persistente existentes do dispositivo móvel. Desta forma temos a certeza que toda a informação que pode ser capturada e sobre a qual exista autorização para a sua aquisição pode ser obtida com sucesso e utilizadas posteriormente de acordo com as melhores práticas da cadeia de custódia *(chain of custody)*.

3.6 SEGURANÇA DE REDE

Além da segurança existente sobre os dados em repouso, o *Android* também dispõe de mecanismos de segurança de rede para proteger os dados em transito, a partir e com destino ao dispositivo *Android*. Seguidamente, esses mecanismos são brevemente analisados.

3.6.1 *DNS sobre TLS*

Para *Android* 9.0 e posterior é incluído o suporte de Domain Name System (DNS) sobre Transport Layer Security (TLS), designado por DNS over TLS (DoT). Os utilizadores dispõem do modo de DNS privado, que permite proteger as consultas ao DNS ao definir um provedor de DNS privado, através da opção presente nas definições de rede. O *Android* 10 vai mais além ao permitir a administradores definirem as suas próprias configurações de DoT. O que permite uma mais fácil monitorização e bloqueio de consultas DNS. Previne também que utilizadores possam alterar as definições de DNS expondo inadvertidamente os pedidos DNS efetuados.

3.6.2 *TLS por omissão*

A partir de *Android* 9.0 é negado todo o tráfego que circule em *clear text*. Apenas domínios especificamente permitidos pelos programadores das respetivas aplicações podem utilizar tráfego não encriptado.

Para prevenir ligações de tráfego não encriptado é necessário incluir no `manifest` o atributo `android:usesCleartextTraffic=["true "false"]`, o qual permite claramente indicar se a aplicação pretende ou não utilizar tráfego encriptado. A opção por omissão desde o *Android* 10 tem o valor de "falso".

Também desde o *Android* 10 é utilizado TLS 1.3 por omissão para todas as ligações por TLS, a qual apresenta melhor desempenho e um aumento da segurança em relação ao seu sucessor. O TLS v1.3 apresenta ainda melhor privacidade ao encriptar melhor o processo de *handshake* e oferece maior segurança ao recusar certificados assinados com algoritmos de `hash` considerados fracos como o `SHA1`.

3.6.3 *Wi-Fi*

Uma grande conjunto de atualizações relacionadas com a segurança foram sendo introduzidas ao longo do tempo, para melhorar a segurança da tecnologia *Wi-Fi*. Sendo esta uma das tecnologias mais utilizadas pela maioria das aplicações móveis é importante perceber as suas limitações no contexto de segurança. Podemos apresentar duas das formas mais utilizadas por atacantes que pretendem explorar vulnerabilidades de uma rede *Wi-Fi*. Apesar das novas normas terem resolvido em grande parte as situações apresentadas, devem ainda ser consideradas pois nem todas as redes ou dispositivos utilizam as normas mais recentes.

Uma das formas pretende-se com um ataque de MITM[8], para isso é criado um *gateway* falso, ao qual o atacante força o cliente a se desconectar da ligação atual, redirecionando para um *rogue gateway*. Desta forma, o atacante consegue ler a informação trocada pelo(s) dispositivo(s) móvel(is) do(s) cliente(s) a este ligado. Por último podemos referir outro ataque, este *offline* que utiliza a informação de *handshake* trocado entre cliente e o *gateway* de acesso, com o intuito de se obter a chave a acesso à rede, conhecida como Pre-shared key (PSK). Este tipo de ataque

8 Existem diferentes técnicas para realizar ataques de MITM, em constante alteração, bem como uma nova nomenclatura designada por Adversary-in-the-Middle (AiTM), mais informações, disponíveis em: https://attack.mitre.org/techniques/T1557/ - Acedido a 18 Mar. 2024.

utiliza *rainbow tables*[9], ou dicionários de palavras como forma de testar todas as combinações possíveis. Este tipo de ataque é mais exigente a nível computacional, no entanto é mais discreto uma vez que todas as tentativas são testadas sem a necessidade de validação com o *gateway* remoto, até que seja encontrada a correta PSK.

O *Android* 10 e superior suporta a norma *Wi-Fi Alliance's Wi-Fi Protected Access version 3* (Wi-Fi Protected Access (WPA)3) e *Wi-Fi Enhanced Open*, ou seja, as normas mais avançadas e recentes aquando a escrita deste documento. Tal permite melhorar a segurança *Wi-Fi* de uma forma geral através da proteção contra os ataques mais comuns conhecidos.

O WPA3 pretende ser uma norma que permita tanto o uso empresarial como doméstico/pessoal, recorrendo ao uso de algoritmos de segurança modernos. *WPA3-Enterprise* utiliza os melhores protocolos e algoritmos de segurança, tais como Simultaneous Authentication of Equals (SAE) com a melhor proteção contra ataques. Já a versão WPA-*Personal* utiliza uma versão simplificada da SAE substituindo o uso de PSK, o que permite que os utilizadores possam ter uma melhor segurança contra ataques conhecidos como *offline dictionary attacks, key recovery* e *message forging*.

O *Wi-Fi Enhanced Open* é a nova norma de segurança desenvolvido pela Wi-Fi Alliance (WFA) para redes públicas, especialmente relevante para dispositivos móveis. É baseada em Opportunistic Wireless Encryption (OWE), esta oferece encriptação e privacidade em redes públicas sem utilização de palavras-chave, onde pode ser encontrada em hotéis, cafés, restaurantes, etc... Note-se que a norma não permite autenticação.

O *Android* também suporta o uso de WPA2-*Enterprise* (802.11i), que é utilizado em redes empresariais, e pode ser integrado com servidores de autenticação remota (RADIUS). Este protocolo suporta o uso do algoritmo `AES-128-CCM`, para autenticação através da encriptação.

9 *How to use precomputed tables to crack Wi-Fi passwords in Hashcat and John the Ripper* - Disponível em: https://miloserdov.org/?p=5167 - Acedido a 17 Mar. 2024

3.7 PROTEÇÃO DE DADOS

3.7.1 *Encriptação*

Encriptação é obrigatória em *Android*, a qual é utilizada para proteger os dados do seu utilizador, na eventualidade de roubo ou perda do dispositivo. O *Android* suporta duas formas de encriptação do dispositivo, File-Based Encryption (FBE) que começou por aparecer a partir do *Android* 7 (API lvl 24), e a Full-Disk Encryption (FDE) que apareceu no *Android* 5 (API lvl 21).

Dispositivos com *Android* 5 até 9 utilizavam FDE, em vez da atual encriptação por FBE. O modo de funcionamento era encriptar todos os dados do dispositivo com uma única chave de encriptação. Neste modo, todos os dados do utilizador são automaticamente encriptados antes de serem enviados para o disco. Na leitura dos dados, esses são automaticamente desencriptados antes de ser devolvida ao processo que os requisitou.

A encriptação FDE é baseada no modo de encriptação do `Kernel Linux`, designado por `dm-crypt` que funciona com mapeamento de blocos físicos em blocos virtuais. O algoritmo de encriptação é `AES-128` ou `AES-256`, com uma cifra de bloco (CBC) e `ESSIV:SHA256`[10].

O **FBE** permite guardar com diferentes chaves de encriptação, as diversas áreas de armazenamento existentes. Os dispositivos com *Android* 10 ou superior vêm com este modo de encriptação ativo por omissão. Com FBE após o dispositivo efetuar o *boot* e o utilizador desbloquear o ecrã, todas as funcionalidades do equipamento podem ser utilizadas.

Dispositivos que fazem uso de FBE podem usar dois tipos de armazenamento para aplicações:

- **Device Encrypted (DE):** após o *boot* do dispositivo, e antes do utilizador desbloquear o mesmo, é disponibilizado o acesso ao disco. Este acesso está protegido por um segredo em hardware e software executados pelo TEE que verifica se o *Verified Boot* foi realizado com sucesso antes de desencriptar os dados.
- **Credential Encrypted (CE):** após o *boot* do dispositivo, e depois do utilizador desbloquear o dispositivo é permitido acesso ao disco. Além das proteções

10 Full-disk encryption - Fonte: https://source.android.com/docs/security/features/encryption/full-disk- Acedido a 30 Maio 2023

utilizadas pelo Device Encrypted (DE), as chaves Credential Encrypted (CE) são derivadas após o desbloqueio do dispositivo, acrescentando assim proteções de hardware contra ataques de força bruta.

3.8 SEGURANÇA DAS APLICAÇÕES

A segurança das aplicações é uma parte integrante e fundamental de uma plataforma móvel, na qual os seus utilizadores confiam tanto para a execução de tarefas, como para comunicação e o armazenamento de dados.

Sendo o estudo da segurança das aplicações o objetivo principal deste trabalho, é importante perceber os mecanismos de segurança existentes no desenvolvimento de aplicações contra *malware*, *exploits* e vulnerabilidades.

Mesmo antes de instalar uma aplicação, é efetuada a procura pela existência de *malware* e outras ameaças que ponham em risco o dispositivo. Essa verificação é efetuada em conjunto com a *Google Play Store* através de *Google Play Protect* e da API da *SafetyNet*. À data de escrita deste documento, a API da *SafetyNet* está a ser descontinuada e por sua vez está a ser substituída pela API da *Play Integrity*[11].

3.8.1 *Application Signing*

O *Android* obriga todas as aplicações a serem assinadas digitalmente com uma `developer key`, antes da sua instalação[12]. As aplicações que tentem instalar sem estarem assinadas são rejeitadas seja através do *Google Play* ou de forma manual pelo instalador de pacote existente no dispositivo *Android*.

O *Android* 9 e superior permitem o uso de uma chave diferente da original/anterior para assinar o APK (*APK key rotation*). No entanto, tem que existir uma forma de relacionar a nova chave com a anterior, permitindo assim a uma nova atualização da aplicação ser assinada com uma chave diferente da original.

Para garantir a integridade e autenticidade do conteúdo da aplicação é utilizado um certificado para identificar o seu criador. Sempre que existe uma instalação ou atualização é feita a comparação com o certificado que assinou o APK em relação ao

11 Play Integrity API - Disponível em: https://developer.android.com/training/safetynet/deprecation-timeline - Acedido a 29 de Jun. 2023

12 Google Docs, assinar aplicações - Disponível em: https://source.android.com/docs/security/features/apksigning - Acedido a 10 de Nov. 2023

anterior no caso de uma atualização. Após a confirmação da validade do certificado é permitida a instalação ou atualização da aplicação.

O *Android* permite também a aplicações (diferentes) assinadas com a mesma chave correr no mesmo processo, se assim for requerido pela aplicação. Esta opção é possível através da referência no `manifest`, através da `sharedUserId`.

É também possível em *Android* através das permissões (`"signature-based"`) expor uma funcionalidade que pode ser utilizada por outra aplicação que tenha sido assinada com a mesma chave. São estas as duas formas de utilizar a mesma chave para assinar a aplicação, que permitem partilhar código e dados de uma forma segura. Mais informação sobre o tópico de permissões, no qual é descrito em pormenor as suas funcionalidades vai ser apresentada na subsecção 3.8.2.

3.8.2 *Permissões*

As permissões em *Android* seguem o modelo de menor privilégio, onde tipos específicos de funcionalidades são apenas atribuídos se forem requeridos pela aplicação.

Existem dois tipos de permissões principais:

- Permissões declaradas pelo `manifest`;
- Permissões declaradas em *RunTime*.

As permissões podem ser requeridas de forma dinâmica (*RunTime* - desde API nível 23, *Android* 6.0) durante o uso da aplicação ao seu utilizador, alertando-o e validando se o mesmo permite ou não a permissão requerida. Este novo modelo de permissões permite ao utilizador ter um maior controlo sobre as funcionalidades da aplicação e os serviços que estão a ser executados (Talegaon e Krishnan, 2020). No entanto estas têm que já estar definidas em conjunto com as restantes no ficheiro *AndroidManifest.xml*. O utilizador pode assim, por exemplo, negar permissões que pretendam ter acesso à sua localização, não negando no entanto à aplicação o acesso à Internet. Para um analista forense, a quantidade de informação possível de ser extraída de um dispositivo móvel não depende apenas do dispositivo, aplicações instaladas, mas também das permissões que foram autorizadas pelo seu utilizador.

Durante a instalação da aplicação, é solicitado ao utilizador a autorização para permitir o acesso a diferentes recursos do dispositivo. Estas permissões estão também definidas no ficheiro *AndroidManifest.xml*, através do elemento *<uses-permission>*.

Quando existem permissões expostas para fora do seu contexto por parte de uma aplicação, é possível limitar o acesso a aplicações com permissões específicas. Permissões são *strings* que identificam a capacidade da aplicação desempenhar uma ação específica. As permissões podem ser mapeadas para grupos de identificação (*Group IDs*) em Linux, que ajudam o *Kernel* a atribuir permissões de acesso a recursos do sistema (Elenkov, 2015).

Uma permissão é composta por três elementos:

1. **Nome** da permissão;
2. **Grupo** da permissão correspondente, de forma a agrupar permissões relacionadas.
3. **Nível** de proteção, indica o impacto que as mesmas podem ter na segurança do dispositivo:
    a) **Normal *(standard)*:** Não causam ameaças conhecidas para a aplicação. As permissões do tipo normal são automaticamente garantidas sem que seja necessário autorização por parte do seu utilizador. Estas permissões requerem acesso a dados ou recursos fora da *sandbox* na qual é executada a aplicação. No entanto existe pouco ou nenhum risco de privacidade para o seu utilizador, ou para a operação de outras aplicações.
    b) **Perigosas *(dangerous)*:** Necessita de elevação de privilégio. O utilizador precisa de aprovar a mesma. É necessária para aplicações ou serviços que requerem dados privados do seu utilizador, ou o qual pode potencialmente impactar a informação guardada do seu utilizador ou presente em "activity"de outras Apps.
    c) **Assinatura:** Apenas aplicações que são assinadas pelo mesmo certificado sobre o componente externo que pretende obter permissão. Este é o tipo mais forte de proteção. Este é usado por programadores para mover recursos e dados entre aplicações do mesmo *developer*. Protegendo assim o seu acesso por parte de aplicações desenvolvidas por outros *developers*.
    d) **Assinatura ou Sistema *(SignatureOrSystem)*:** Este tipo de permissão não é recomendado. São por norma atribuídas a aplicações que fazem parte da imagem do sistema, ou que são assinadas com a mesma chave/certificado que declarou as permissões (Elenkov, 2015).

As permissões com o nível de proteção de *SignatureOrSystem*, são utilizadas com aplicações que vêm pré-instaladas num dispositivo *Android*. Estas aplicações geralmente encontram-se no diretório */system/app* ou */system/priv-app*. As apli-

cações que tem este tipo de permissão devem ser verificadas, a razão sendo que muitas destas têm demasiadas permissões (permissões elevadas/`root`). Exemplos encontram-se em aplicações produzidas pelo Android Open Source Project (AOSP), pelo fabricante do dispositivo móvel ou pelo operador de rede móvel.

Em *Android* cada aplicação está restringida a uma *sandbox* que necessita de permissões específicas para permitir a interação entre aplicações ou com o sistema. Serviços de alto nível do sistema obrigam o uso de permissões de forma a obter o UID da aplicação que executa a chamada, através do uso *Binder*.

3.9 COMPONENTES DAS APLICAÇÕES

3.9.1 *Activities*

As atividades ou *Activities* (termo anglo-saxónico) correspondem ao ponto inicial de entrada para uma aplicação *Android*. A cada atividade corresponde um único ecrã que se encontra em execução e o qual está a ser mostrado (*foreground*) pelo dispositivo móvel. Podemos afirmar que uma aplicação é composta por múltiplas *Activities* interligadas entre si, as quais definem a forma como o utilizador interage com a mesma. Uma atividade pode ser implementada ao estender a classe *Activity*. Um utilizador pode também lançar uma atividade de outra aplicação através de um *Intent*.

Todas as *Activities* presentes numa aplicação precisam de ser declaradas no ficheiro `manifest`. Qualquer *activity* que não esteja declarada no `manifest` não será registada no sistema e como tal não é permitida a sua execução (Anmol Misra, 2016). Uma vez que uma aplicação pode começar *activities* com outras aplicações é importante controlar o seu uso. Para tal devem ser definidas permissões no ficheiro `manifest`, exigindo assim que outras aplicações tenham que requerer permissão através da instrução *uses-permission*. Se a *activity* que fizer a chamada não tiver a devida permissão, o pedido para começar uma nova *activity* é negado.

3.9.2 *Intents*

Os *Intents* representam um mecanismo fundamental em *Android* de comunicação entre processos (IPC). Estes são o meio principal para aplicações comunicarem e/ou interagirem entre componentes (*activities, services e broadcast receivers*), ou entre

outras aplicações, mesmo que o destinatário não esteja em funcionamento. As tarefas mais frequentes, tais como clicar numa hiperligação para iniciar o browser, receber uma notificação a indicar que uma nova SMS foi recebida, ou instalar/remover aplicações. Todas estas ações envolvem a troca de *Intents* por todo o sistema.

Os *Intent's* podem ser usados para:

- Iniciar uma *activity*, ativando a interface de utilizador de uma aplicação;
- Enviar *broadcasts* para informar o sistema ou aplicações de alterações;
- Para iniciar, terminar e comunicar com um serviço em *background*;
- Aceder a dados através de *Content Providers*;
- Como chamadas para lidar com eventos.

A incorreta implementação pode resultar em perda de dados, restrição de funções a serem chamadas e manipulação indevida do fluxo da aplicação.

Implicit Intent
Um `implicit intent` é utilizado para comunicar entre duas ou mais `activies` em diferentes aplicações. O que o distingue é não ser necessário indicar qual o componente a que se destina, apenas a ação a executar.

Explicit Intent
Por outro lado o `explicit intent` funciona para dentro da aplicação. Por exemplo a comunicação entre duas `activities`, ou serviço da mesma aplicação. Ao utilizar um `explicit intent` é necessário especificar o nome da classe ao qual se destina.

Intent-Filter
Um `intent-filter` define os tipos de `Intents` que uma atividade, serviço ou *broadcast receiver* pode responder. Este também especifica o que a atividade ou serviço pode fazer, e quais os tipos de *Broadcasts* um *Receiver* pode lidar. Podemos identificar os componentes que podem receber *Intents* de acordo com o tipo declarado.

Os `intent-filter` estão especificados no `AndroidManifest.xml`, no entanto para *Broadcast Receivers* é também possível definir o mesmo no código. Ao definirmos podemos então especificar a sua categoria, ação, dados e metadados.

Um `intent-filter` é sempre componente público, mesmo que não seja especificado `android:exported="true"`, no entanto estes podem ser alterados pelos programadores de forma individual, para cada componente através de `android:exported="false"` no `manifest`. Ao definir `android:exported="false"`, estamos a limitar o acesso

do componente apenas à própria aplicação no qual este foi definido. Ou através da definição de uma **permissão** de acesso mais restrita ao componente.

3.9.3 *Serviços*

Um serviço em *Android* é prestado através de uma funcionalidade que está a correr em *background* (ex: processamento de dados, iniciar *intents* e notificações, etc.), mesmo quando a aplicação não está a ser executada em primeiro plano (*foreground*). Serviços foram projetados para processos que são executados a longo tempo. A sua prioridade é mais baixa que aplicações em execução, no entanto é mais elevada do que aplicações que se encontrem inativas. Desta forma a probabilidade do sistema terminar a sua execução é mais baixa, se existir a necessidade de obter mais recursos do sistema, podendo ser novamente restaurados quando os recursos voltarem a estar disponíveis.

Os serviços, assim como os encontrados em *Activities*, são executados na *thread* da aplicação principal. Portanto, um serviço não inicia a sua própria *thread* e não pode ser executado em um processo separado, a menos que seja explicitamente definido de outra forma.

A tecnologia utilizada para correr tarefas associadas a estes serviços, sofreu alterações ao longo dos tempos. A partir de *Android 5.0 (API level 21)* e sucessivas implementações, pode ser iniciado um serviço através de *JobSchedulers*.

3.9.4 *Broadcast Receivers*

Os *Broadcasts* são utilizados como um sistema de envio de mensagens, sendo os *Broadcasts Receivers* componentes que aguardam o envio dessas mensagens. Se uma aplicação registar um *receiver* para um tipo específico de *broadcast*, o código desse *receiver* só será executado quando o sistema enviar a mensagem (*broadcast*). Neste caso, a mesma mensagem pode ser recebida em diversas aplicações.

Existem duas formas de se declararem um *receiver*, seja através do `Manifest` ou registada de forma dinâmica no código da aplicação, através da chamada à API `registerReceiver()`.

Os *broadcast receivers* são assim o único componente que pode ser definido no código fonte (Java) sem existir o seu registo no *Manifest*. Esta característica deve-se à função dos *broadcast receiver*, os quais são utilizados para lidar com eventos que

são gerados, tanto pelo sistema como por outras aplicações. Essa característica permite uma maior flexibilidade ao programadores para lidar de forma dinâmica com os eventos ocorridos, sem necessitar de declarar os mesmos no `Manifest`.

3.9.5 *Content Providers*

Content providers, ou provedores de serviço são utilizados para gerir partes de dados da aplicação para que estas possam ser partilhadas com outras aplicações no mesmo dispositivo. Ao utilizar um Uniform Resource Identifier (URI) outras aplicações podem inquirir ou modificar os dados, mesmo que este não pertença à aplicação em execução. O uso de *Content Providers* podem ser encontrados em imagens, ficheiros de texto, base de dados Structured Query Language (SQL), etc. É importante validar os dados recebidos, com o intuito de evitar vulnerabilidades, tais como *SQL Injection*.

Uma vez que estes são utilizados para trocar informação é muito importante definir permissões com o nível correto de proteção. Estes utilizam os atributos `readPermission e writePermission`, para definirmos quais as permissões que a aplicação deve ter. Este tipo de permissões tem precedência sobre o atributo `<permission>`, utilizado no `Manifest`.

As principais características dos *Content Providers* são:

- Permite a troca de dados entre diferentes aplicações após um pedido;
- Permite guardar dados da aplicação em diferentes meios de armazenamento (sistema de ficheiros, base de dados, web, etc..), desde que tenha o devido acesso;
- Poder ser usado por outras aplicações para obter ou modificar os dados (se o *Content Provider* o permitir), conforme necessário.
- Útil se for necessário partilhar dados com outras aplicações;
- Uso similar ao de uma BD, com os métodos:
  - `insert()`;
  - `update()`;
  - `delete()`;
  - `query()`;

Content Resolver

Este é responsável por estabelecer o acesso aos dados presentes no *Content Provider*, é através do objeto `ContentResolver`[13] no contexto da aplicação que permite a comunicação como cliente ao *provider*.

O objeto `ContentResolver` comunica com o objeto *provider*, que é uma instância da classe que implementa o `ContentProvider`. Desta forma percebemos que o *Content Resolver* é o responsável por estabelecer a comunicação com o *Content Provider*, que por sua vez é o responsável por aceder e devolver os dados armazenados. A figura 6 (página 51) ilustra a interação do *Content Resolver* com o *Content Provider*, a base de dados e as restantes classes.

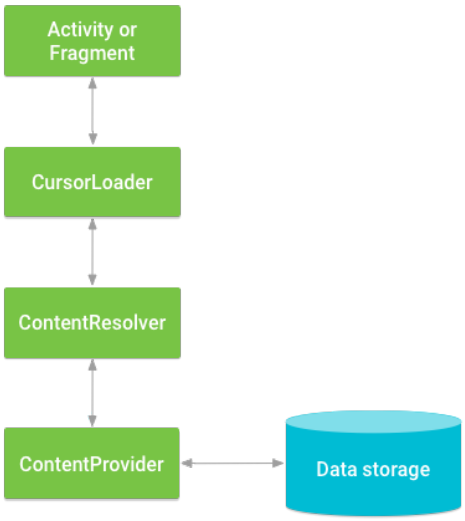


Figura 6: Interação Content Provider e Resolver

Fonte: https://developer.android.com/guide/topics/providers/content-provider-basics

3.10 ESTRUTURA DE UM FICHEIRO APK

É fundamental perceber a estrutura dos ficheiros existente num APK, o qual nada mais é que arquivo *zip*. Esta secção serve para especificar o conteúdo e finalidade da estrutura de ficheiros que encontramos ao descompilar um ficheiro APK. Podemos observar através da figura 7 os diferentes diretórios e ficheiros que fazem parte de um APK.

13 Android Developer, content provider - Disponível em: https://developer.android.com/guide/topics/providers/content-provider-basics - Acedido a 16 Mai. 2023

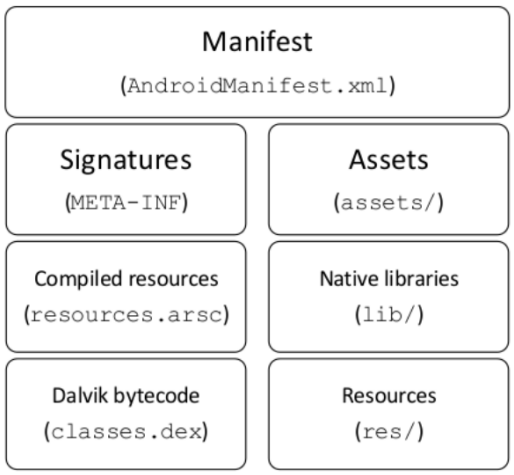


Figura 7: Estrutura de ficheiros de um APK

Fonte: (Ratazzi, 2016)

Um dos ficheiros mais importantes presentes no APK é o `AndroidManifest.xml`, já por várias vezes referido neste documento. Este ficheiro Extensible Markup Language (XML) é responsável por descrever ao sistema o comportamento dos componentes da aplicação *(activities, services, broadcast receivers e content providers)*. Este ficheiro também especifica todas as permissões que são necessárias, conforme já observado na secção 3.8.2.

O ficheiro `Classes.dex` contém o código executável da aplicação no seu formato nativo DEX, que pode ser executado em DVM ou ART. Aplicações em *Android* estão limitadas no número de métodos por ficheiro DEX. Quando uma aplicação ultrapassa esse limite, durante o processo de construção da aplicação é feita a divisão do código em múltiplos ficheiros DEX, esta técnica é conhecida como *"multidexing"*. O mesmo pode ocorrer se uma aplicação utilizar bibliotecas ou dependências externas que têm o seu próprio ficheiro DEX.

A diretoria `resources.arsc` contém todos os recursos compilados da aplicação, tais como strings e estilos utilizados.

A diretoria `assets` pode conter ficheiros em formato não compactado (`raw`), bem como ficheiros de música, imagens, formatação (`fonts`).

As aplicações podem tirar proveito do conjunto de bibliotecas através da Java Native Interface (JNI), presentes na diretoria `lib`. Esta dispõem de bibliotecas, organizadas por subdiretorias para as diferentes arquiteturas do SO.

Os recursos que são diretamente referenciados a partir do código *Android*, ou indiretamente através de API's de alto nível, localizados na diretoria `res`.

Tal como sucede em ficheiros *JAR*, num APK está também presente a diretoria `META-INF`, a qual contém ficheiros de assinaturas, são estes o `"MANIFEST.MF", CERT.RSA, CERT.SF`.

A implementação da assinatura de código em Java é efetuada com a junção de outro ficheiro do `Manifest`, designado por ficheiro de assinatura, o qual tem a extensão ".SF". Este contém os dados a serem assinados e é assinado de forma digital sobre si mesmo. Esta assinatura digital também chamado por *"signature block file"* é guardado em conjunto com um ficheiro binário, com a extensão `.RSA, .DSA` ou `.EC`, conforme o algoritmo de assinatura utilizado.

3.10.1 *Classes.dex*

Tal como referido anteriormente após a descompilação do APK é obtido o `.dex`. Este ficheiro é utilizado pela máquina virtual para correr o `bytecode`, seja esta em DVM ou ART.

Na perspetiva de *reverse engineer*, o nosso objetivo é a partir do *bytecode dex*, converter para *SMALI*, e por fim obter o código Java.

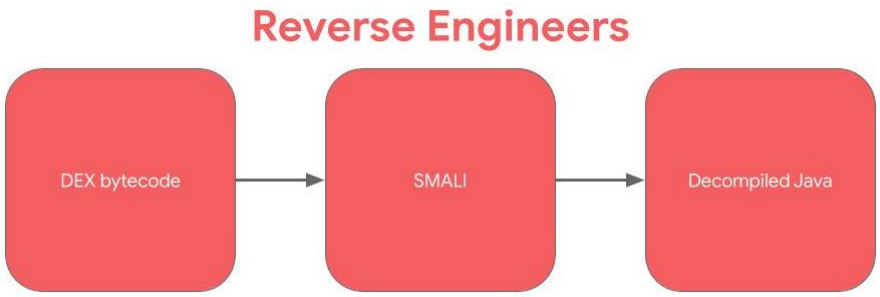


Figura 8: Processo de Engenharia Reversa

Fonte: https://www.ragingrock.com/AndroidAppRE/app_fundamentals.html

Como este código é linguagem máquina é necessário utilizar ferramentas que nos permitam perceber o código que será utilizado para correr a aplicação. As ferramentas utilizadas estão descritas na secção 4.2.

3.10.2 *Manifest*

O ficheiro `manifest` de uma aplicação é criado antes da sua compilação e não pode ser alterado durante o seu tempo de execução. A forma como o seu con-

teúdo é apresentado é semelhante a XML. O ficheiro resultante é designado por `AndroidManifest.xml`. Estes detalham todos os componentes descritos anteriormente (Atividades, *broadcast receivers*, serviços, etc.). O ficheiro `manifest` também detalha as permissões necessárias pela aplicação, tal como o nível da API mínimo referente à versão *Android*, bem como as necessidades a nível de hardware e software da aplicação.

Apesar de existir apenas **um único** ficheiro `manifest`, se existirem bibliotecas importadas, as mesmas podem ter o seu próprio ficheiro `manifest`. Revelando assim um dos problemas existentes em pequenas empresas de desenvolvimento de aplicações para *Android*. Ao utilizar bibliotecas de entidades não confiáveis, podemos estar a exigir permissões em excesso, desnecessárias para o funcionamento pretendido da aplicação.

Devido a esta situação, durante a construção da aplicação o *Gradle* junta todos os ficheiros `manifest` das diferentes bibliotecas (se estes existirem) num único `AndroidManifest.xml`[14]. Ao realizar técnicas de engenharia reversa, deve ser possível observar o tamanho acrescido no ficheiro `AndroidManifest.xml` final e os respetivos elementos adicionais, em relação ao original.

3.11 CERTIFICATE PINNING

O processo de *certificate pinning* reside na associação do servidor (*host*) ao certificado x509 ou chave pública designada. Após a chave pública ou certificado ser conhecido é feita a sua associação (*"pinned"*) ao servidor (*host*). Pode no entanto existir mais que um certificado ou chave pública atribuído a um *host*, sendo mantido este conjunto num *"pinset"*. Desta forma a aplicação não irá confiar em certificados modificados pelo utilizador ou outra entidade que não a aceite previamente e também não irá permitir que ferramentas de *proxy* interceptem e consequentemente decifrem o tráfego.

Apesar do conceito de *certificate pinning* não ser limitado a DM, o seu uso é comum em aplicações móveis. Este foi criado para melhorar a segurança dos canais de comunicação, uma vez que os existentes não correspondiam as exigências dos seus utilizadores. Os quais nos referimos são VPN, Secure Sockets Layer (SSL) e TLS, estando estes vulneráveis a um conjunto de ataques conhecidos[15].

14 "Merge multiple manifest files": https://developer.android.com/build/manage-manifests - Acedido a 11 Nov de 2023.

15 HeartBleed - Disponível em: https://www.troyhunt.com/everything-you-need-to-know-about3/ - Acedido a 29 de Jul. 2023

O conceito de *pinning* funciona com base no conhecimento à priori entre o utilizador e a organização ou serviço que pretende utilizar. Isto significa que não é necessário utilizar mecanismos "genéricos"para resolver o problema na troca de chave que pode ocorrer. Ou seja não é necessário utilizar DNS para resolução de endereços/nomes ou Certificate Authority (CA)'s para estabelecer o estado ou validade do certificado. Existem no entanto falhas conhecidas no sistema de *pinning*[16].

Um servidor ou serviço de certificados ou chaves públicas pode ser adicionado a uma aplicação durante o seu desenvolvimento, ou pode ser adicionado quando a aplicação tenta pela primeira vez a ligação, processo este conhecido como *key continuity*. No entanto o processo mais seguro é a associação no desenvolvimento, uma vez que limita a possibilidade de um atacante forjar a associação.

Em DM que utilizam *Android* N (7.0) ou superior, a implementação é feita através da funcionalidade do *Android* conhecida por *Network Security Configuration*[17]. Este recurso permite às aplicações declarar as suas configurações num ficheiro de forma isolada, sem ser necessário alterar o código existente da aplicação.

Ao utilizar o *certificate pinning*, existe por norma uma *backup key*, na eventualidade de existir necessidade de alterar ou criar uma CA's, desta forma é evitada a ocorrência de falhas na conexão (remota) da aplicação[18].

No entanto é importante destacar que se não existirem outras medidas, tais como a deteção de *root*, ofuscação, *anti-tamper* e proteções execução/binárias a implementação de *certificate pinning* será insuficiente. Quando uma aplicação com um certificado de *pinning*, permite ser executada em um DM com privilégios de *root*, um atacante pode facilmente aceder aos dados encriptados através de técnicas de análise dinâmica, e assim ultrapassar o *pinning* estabelecido. Esta técnica é empregue neste trabalho na secção 7.1.3.

POODLE - Disponível em: https://www.troyhunt.com/everything-you-need-to-know-about/ - Acedido a 29 de Jul. 2023
VPN, Mitre CVE - Disponível em: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=vpn - Acedido a 29 de Jul. 2023

16 Pinning Gaps - Disponível em: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning#Pinning_Gaps - Acedido a 30 de Jul. 2023

17 Network Security Configuration, Android Docs - Disponível em: https://developer.android.com/training/articles/security-config?hl=pt-br - Acedido a 30 de Jul. 2023

18 Certificados Pin - Disponível em: https://developer.android.com/training/articles/security-config.html#CertificatePinning - Acedido a 30 de Jul. 2023

4

AMBIENTE DE TESTES

Antes mesmo de começar a analisar os artefactos gerados pela aplicação é necessário definir quais as ferramentas a serem utilizadas. É ainda importante configurar as mesmas, percebendo quais as suas capacidades e nuances de forma tirar o melhor partido das mesmas.

Por forma a executar os diferentes tipos de análise podemos recorrer a dois tipos de dispositivos, emulados (virtuais) ou reais (físicos). Foi descartada a opção de recorrer a um serviço de virtualização de *Android* através de *container*, pelas razões que são apresentadas mais a frente neste capítulo. Os serviços de virtualização oferecem várias vantagens, especialmente no dimensionamento da quantidade de aplicações a serem testadas em comparação com emuladores e/ou dispositivos físicos. Além disso, os serviços de virtualização destacam-se ao lidar com limitações de hardware, proporcionando mecanismos que facilitam a recuperação do ambiente de testes desejado, ao contrário de dispositivos físicos (Chau e Jung, 2018).

Apesar das vantagens que foram apresentadas anteriormente existiram três razões principais que levaram à escolha de um DM físico para a realização deste trabalho. A necessidade de utilizar um dispositivo com localização em tempo real (GPS), a exigência quanto à portabilidade para testar a aplicação na forma mais aproximada ao uso em contexto real, bem como a singularidade da aplicação em análise, a qual exige interacção em tempo real entre o utilizador e a aplicação em execução no DM.

Para que não existam restrições durante a manipulação do DM, existiu a necessidade de utilizar um DM físico com privilégios de *root*.

- **Dispositivo físico:**
  Foi apenas utilizado um único dispositivo durante toda a fase de testes, e as suas características são apresentadas na Tabela 5 (página 58).
- **Especificações Android:**
  No início do trabalho foram realizadas diversas tentativas com versões mais antigas de *Android*. O uso de versões *Android* mais antigas pode ser vantajoso, com o intuito de facilitar a descoberta de vulnerabilidades. A razão

Tabela 5: Características do dispositivo móvel de testes

Marca	Xiaomi
Modelo	Redmi Note 3
CPU	Snapdragon (Qualcomm Technologies, Inc MSM8956)
Codename	Kenzo
Arquitetura	Armv8 - aarch64
Versão Firmware	V8.2.1.0.MHOCNDL
Versão Kernel	*Linux version 3.10.108 Escrima X28 Kamui LightningBlade Purple-Lightning*

de tal acontecer depreende-se com diversos critérios. A existência de menos mecanismos de segurança, os quais foram sendo acrescentados a cada nova versão do *Android*. Acresce-se a inexistência de atualizações de segurança, por parte dos fabricantes de DM, o que culmina na existência de vulnerabilidades bem conhecidas e fáceis de explorar, sobre as quais não existem *patchs* de segurança. Os testes iniciaram-se com a versão *Android* 5.1 a qual correspondia à versão original do dispositivo de testes. Durante a fase de *root*, além de se ter realizado um *factory reset* ao dispositivo foi também feita a sua atualização para a versão 6.0 (*Marshmallow*), a qual corresponde à ultima versão com suporte oficial pelo produtor do DM.

No entanto problemas de compatibilidade com diversas ferramentas, com maior destaque pela *Frida*, levaram à escolha de uma *custom ROM*. Esta ROM para dispositivos *Android* é baseada no AOSP, mantida pela Google e é usada como base para diversas ROM modificadas incluindo a *"Pixel Experience"*, sendo esta a escolha utilizada neste trabalho. A versão da API *Android* 28, é outra das características que afetam o funcionamento da aplicação, a escolha a razão desta escolha foi devido a compatibilidade com as ferramentas *MITM proxy* e *Frida*.

A tabela 6 apresenta um resumo das características mais relevantes do SO *Android* que foi utilizado no dispositivo de testes.

- **Aplicação Bolt:** A versão da aplicação *Bolt* escolhida diz respeito à ultima versão disponível à data de início deste trabalho. Como tal não foi escolhida ou tido em conta falhas ou bugs conhecidos à priori. Mais informações sobre a aplicação em estudo encontram-se na tabela 7.

Tabela 6: Características do sistema operativo do dispositivo de testes

Característica	**Especificação**
Versão Android	9.0 (Pie)
Versão Software Development Kit (SDK)	28
Android Operative System (OS)	*PixelExperience* (AOSP)
Versão OS	`20190906-0737-OFFICIAL`
OS Build	`PQ3A.190801.002`

Tabela 7: Detalhes da aplicação *Bolt Request a Ride*

Designação	Bolt *Request a Ride*
Versão	CA.52.1
Data de lançamento	7 Novembro de 2022
Application Build	2086
Tamanho	61.6 MB
MD5	`200be9ffcb21a723703038dfefbebec4`
SHA-256	`3bc581327e5353830b03dba87efe9fa60afce9e774a027488edd345bda9e36fe`

4.0.1 *Root Dispositivo Móvel*

O processo de *rooting* do DM é fundamental para ser possível ultrapassar as limitações impostas por fabricantes de dispositivos em *Android*. É considerado que temos um dispositivo *rooted*, quando temos acesso a uma conta de administrador de sistema, que tenha acesso de 'Super Utilizador' (`root account`).

Um dispositivo *rooted* permite alterar ou substituir não apenas aplicações ou definições de fábrica pré-existentes no dispositivo, mas também executar aplicações ou realizar operações que necessitam de permissões de administrador. Acresce-se que *root* num dispositivo permite o acesso aos dados privados da aplicação, se esses não estiverem cifrados pela própria aplicação.

Na perspetiva de uma analista de segurança ou forense, a necessidade de obter *root* prende-se com o uso e acesso a partes do sistema que não é possível aceder por parte de um utilizador comum.

Para realizar o *root* ao dispositivo móvel as ferramenta necessárias foram, Android Debug Bridge (ADB) e *fastboot* disponíveis em *Android SDK Tools*[1], e a ferramenta

1 Android SDK Tools - Disponível em: https://developer.android.com/tools - Acedido a 6 Out. 2022

de `root Magisk`[2]. Para o processo de *flash*, foi empregue a ferramenta de código aberto Team Win Recovery Project (TWRP)[3]. Estas ferramentas são as mais sólidas e com um histórico bem sucedido para diversos dispositivos e `ROMs`. Existindo outras soluções para atingir o mesmo objetivo a escolha foi, devido não só ao seu fácil e genérico processo de `rooting`, mas também à forma como é obtido `root` parcial, que nos permite escolher quais as aplicações que podem ter acesso de `root`.

A necessidade de aceder aos dados privados de cada aplicação presentes em `/data/data`, é também de extrema relevância no contexto forense. No entanto efetuar *rooting* a um dispositivo móvel traz diversas consequências que podem ser negativas, se não forem tidos os procedimentos corretos, podendo dar origem a consequências dramáticas para o seu utilizador.

- **Risco Segurança:** Mais exposto a riscos de segurança, uma aplicação maliciosa pode ter acesso a todo o sistema operativo e todos os seus dados, bem como aos dados de outras aplicações instaladas no dispositivo.
- **Inutilizar o dispositivo:** Se o processo de *root* for mal realizado pode inutilizar o dispositivo.
- **Perda de garantia:** O fabricante pode considerar que o seu utilizador expôs o dispositivo a um risco desnecessário invalidando a sua garantia.
- **Implicações Forenses:** O acesso a um maior conjunto de funcionalidades e de dados pode levar a alteração indevida de partes de dados, o que põem em causa a validade forense.

Não foi imposta qualquer limitação de uso da aplicação *Bolt*, após o *root* do dispositivo, nem qualquer notificação sobre o mesmo. O que indica que não existem mecanismos que previnam ou limitem as funcionalidades da aplicação ao ser executada sobre um dispositivo com privilégios de *root*.

As ferramentas empregues para análise da aplicação Bolt são apresentadas no próximo capítulo.

2 Github Magisk - Disponível em: https://github.com/topjohnwu/Magisk - Acedido a 10 de Set. 2022

3 TWRP, Xiaomi Redmi Note3 - Disponível em: https://twrp.me/xiaomi/xiaomiredminote3.html - Acedido a 15 de Set. 2022

4.1 FERRAMENTAS DE ANÁLISE A APLICAÇÕES

Esta secção pretende descrever de uma forma breve todas as ferramentas que foram utilizadas para facilitar o processo de análise da aplicação. O capítulo está divido pelas duas formas de análise estática e dinâmica, para que seja mais fácil ao leitor identificar as ferramentas que foram utilizadas para uma e outra análise, sendo que ambas se complementam.

4.2 FERRAMENTAS ANÁLISE ESTÁTICA

4.2.1 *ADB*

O ADB é uma ferramenta de linha de comandos que nos permite comunicar de uma forma fácil com o dispositivo *Android*, seja físico ou virtual.

Diversas ações podem ser desempenhadas recorrendo a esta ferramenta, desde instalação e remoção de aplicações. Uma das ferramentas do ADB designada por Activity Manager (AM), permite efetuar diversas ações sobre o sistema. Desde iniciar uma `activity`, terminar um processo, modificar as propriedades do ecrã, entre muitas outras.

4.2.2 *Apktool*

O *Apktool* é uma das ferramentas fundamentais no processo de engenharia reversa. O *Apktool* permite descompilar o APK de aplicações *Android*, possibilitando a extração do código fonte e todos os recursos que são compilados pelos desenvolvedores da aplicação.

Neste trabalho foi utilizada a versão *Apktool v2.7.0*, sendo esta uma ferramenta *open-source*, pode ser obtida através do repositório oficial do desenvolvedor da ferramenta[4].

Sem esta ferramenta não era possível ter acesso ao código fonte, e a todos os recursos que a mesma necessita para a sua execução, uma vez que não somos os criadores da aplicação. No entanto o resultado final descompilado que foi obtido, resultante do uso da ferramenta *Apktool* não representa uma cópia fiel do código

4 Repositório - Apktool - Disponível em: https://bitbucket.org/iBotPeaches/apktool/downloads/ - Acedido a 10 Mar 2023

fonte original. Isto acontece porque o APK foi criado recorrendo a ferramentas de ofuscação de código, como a *Dexguard, Allori, DashO*[5], etc.. Este tipo de ferramentas utiliza técnicas de ofuscação que limitam o resultado do processo de descompilação, para que atacantes ou competidores de mercado não possam facilmente obter o código fonte original. Será referido ao longo deste documento, as dificuldades e limitações que existiram quanto à clareza da lógica que é utilizada pela aplicação, devido em grande parte ao resultado limitado, que foi obtido por esta ferramenta (*Apktool*).

Além desta funcionalidade, o *Apktool* permite também reconstruir (recompilar) a mesma aplicação, sendo assim uma alternativa útil durante o processo de análise de segurança de uma aplicação. É particularmente relevante quando pretendemos explicitamente alterar o seu funcionamento para estudo do respetivo comportamento.

Podemos assim dizer que esta ferramenta pode ser utilizada durante as duas fases de análise, seja na análise estática ao código e ficheiros locais, como na análise dinâmica com a modificação do código existente.

4.2.3 *Androguard*

O Androguard é uma ferramenta que pode ser utilizada com diferentes finalidades. A ferramenta facilita o processo de engenharia reversa, ao reunir um conjunto de diversas funcionalidades desempenhadas por diferentes ferramentas numa só.

Listagem 1: Androguard: Criar gráfico de controlo de fluxo

```
1    androguard decompile Bolt_52.1.apk -o bolt_fg -f png --limit
     ↪  '^Lee/mtakso/client'
```

O comando mostrado na Listagem 1 é uma expressão das capacidades desta ferramenta. Podemos observar através da figura 9, um exemplo obtido pelo comando da Listagem 1. Ainda na Figura 9 podemos constatar que cada nó representa um método do código em conjunto com as ligações existentes entre estes. É utilizada a sintaxe *SMALI* na descrição do fluxo de chamadas.

No exemplo, a *flag –limit* limita/constringe as classes que se pretende observar. O objetivo é controlar a complexidade da representação visual. Isto para que o gráfico criado não se torne demasiado complexo, impedindo que se possam retirar conclusões.

5 Ferramentas de Obfuscação - Disponíveis em: https://www.guardsquare.com/dexguard, https://allatori.com/, https://www.preemptive.com/products/dasho/ - Acedido a 14 Nov. 2023

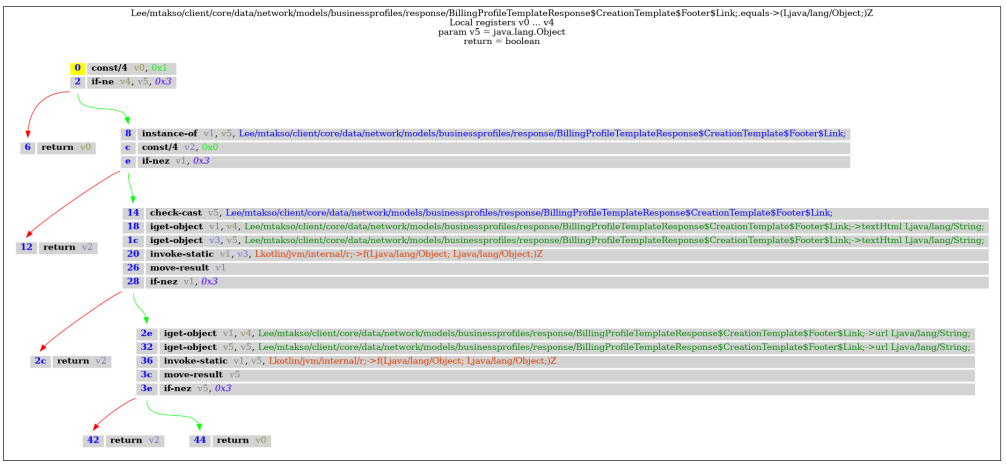


Figura 9: Androguard: Exemplo do gráfico de fluxo

Outra das mais valias desta ferramenta é a criação de gráficos de chamadas, conhecidos por '*call graph*', usando-se para o efeito a *flag 'cg'*[6].
Ambos os comandos podem ser utilizados para se perceber a estrutura da aplicação e como esta opera. Com aplicações de grandes dimensões ferramentas que nos permitam perceber a lógica da aplicação, podem-se tornar fundamentais para descobrir vulnerabilidades e auxiliar durante uma análise dinâmica.

4.2.4 *App Link Verification*

A *App Link Verification* foi desenvolvida por uma investigadora portuguesa[7], a qual permite listar e criar um Proof of Concept (POC), para testar os *deep links* encontrados. Esta ferramenta permite efetuar diferentes modos de operação todas relacionadas com *deep links*.

A mais valia desta ferramenta é que permite efetuar diversas operações todas estas relacionadas com *deep links*. O seu uso é intuitivo e detalhado, permitindo não só obter a lista dos *deep links* existentes como testar os mesmos sobre o dispositivo móvel real ou emulado. Como tal provou ser uma ferramenta útil durante os testes efetuados neste trabalho.

6 Androguard, Call Graph - Disponível em: https://androguard.readthedocs.io/en/latest/tools/androcg.html - Acedido a 15 Nov. 2023
7 App Link Verification, github. Disponível em: https://github.com/inesmartins/Android-App-Link-Verification-Tester - Acedido a 3 de Ago. 2023

4.2.5 *dex2jar*

Esta ferramenta permite converter os ficheiros `classes.dex`, em ficheiros Java Archive (JAR). No *pipeline* do processo de engenharia reversa esta ferramenta está situada entre o processo de descompilação efetuado pelo `Apktool` (capítulo 4.2.2) e a leitura dos ficheiros JAR por uma ferramenta que vai ser apresenta também nesta secção, designada por `Jd-Gui` (capítulo 4.2.7).

Esta é mais uma ferramenta *open-source* e pode ser encontrada na página do seu criador[8].

4.2.6 *Logcat e pidcat*

O `logcat` é uma ferramenta de linha de comandos, utilizada para consultar os registos de mensagens (*logs*) do sistema. Estes registos podem ser desde mensagens de *debug*, informação de eventos do sistema, inclusive mensagens criadas pelos desenvolvedores de aplicações.

Alguns dos *logs* registados pelo `logcat` podem conter informação considerada como confidencial, e que por sua vez não deve constar nos *logs*. A informação confidencial/sensível não deve ser escrita para o `logcat`, podendo existir aplicações terceiras que recolham de forma indevida esta informação e enviem para um servidor externo por parte de um atacante que tenha conseguido estabelecer acesso remoto ao DM.

O uso do *logcat* requer uma filtragem por Process Identificator (PID) da aplicação, de forma a restringir a quantidade de mensagens de `logs` recolhidas, este requisito deve-se à extensa informação que é gerada por DM's. Podemos no entanto efetuar a mesma tarefa, através de uma ferramenta como o `pidcat`[9], onde apenas é necessário referir o nome do *package* da aplicação (`ee.mtakso.client`), conforme mostrado na Listagem 2. O *pidcat* dispõem de uma interface mais ilustrativa e simplista em relação ao *logcat*.

Listagem 2: Ferramenta pidcat: Consultar logs da aplicação Bolt

```
1 pidcat ee.mtakso.client
```

8 Código fonte, Dex2Jar - Disponível em: https://github.com/pxb1988/dex2jar - Acedido a 9 Set. 2022

9 Código fonte, Pidcat - Disponível em: https://github.com/JakeWharton/pidcat - Acedido a 15 Set. 2022

O uso desta pequena ferramenta permite, em tempo real, acompanhar o que está a ser escrito para o *log*.

Da mesma forma ao consultar o código fonte da aplicação é possível identificar o que poderá ser escrito através da pesquisa pelo método `Log.d()`, dado que o mesmo é utilizado para escrever valores de variáveis no processo de *debug*.

Após concluir a fase de procura inicial, o próximo passo consiste na pesquisa por `tokens`, identificadores de sessão (`session ID`) e qualquer outra informação que possa estar a ser escrita para o `logcat`. Este processo permite ampliar a perceção e compreensão do que está a ser executado pelo DM, e procura identificar possíveis dados sensíveis ou muito relevantes para a análise à aplicação móvel.

4.2.7 *jd-gui*

O *jd-gui* é uma ferramenta *open-source*[10] que permite a leitura de ficheiros Java. Devido à sua interface simples, esta é usada apenas para leitura do código fonte, sem qualquer tipo de edição do mesmo ou compilação do código.

4.2.8 *Jadx*

O *Jadx* é uma das ferramentas de código aberto fundamentais e indispensável[11], dado que facilita o processo de engenharia reversa para aplicações móveis. O *Jadx* é de uso fácil e intuitivo, isto porque é apenas necessário indicar a *path* para o ficheiro `.apk` que pretendemos analisar. A ferramenta efetua a descompilação do mesmo, obtendo o `.dex` e procede à sua conversão para *Smali*. Por fim efetua o processo de converter o código *Smali* para Java, e revela ao analista o código obtido das tarefas que foram realizadas.

Esta ferramenta tem uma capacidade limitada para efetuar o processo de desobfuscação, renomeando os métodos das classes com identificadores únicos. No seu processo de desobfuscação, podemos definir o tamanho máximo e mínimo (de caracteres) utilizado para a renomeação de classes, métodos e variáveis.

Mesmo ao desativar a desobfuscação efetuada pelo *Jadx*, este renomeia as variáveis, métodos, funções e classes em todos os casos que existam nomes repetidos. No entanto é mantido em conjunto com o código fonte criado pelo *Jadx* o nome original

10 Jd-Gui Github - Disponível em: http://java-decompiler.github.io/ - Acedido a 18 Dez. 2022
11 Jadx - Disponível em: https://github.com/skylot/jadx - Acedido a 20 Dez. 2022

para cada um dos casos anteriormente referidos. Neste trabalho optou-se pelo uso desta ferramenta sem qualquer técnica de desobfuscação. Isto porque, como iremos ver mais adiante, existe a necessidade de utilizar os nomes originais para que seja possível a sua manipulação durante a análise dinâmica.

4.2.9 *sqlite3-android*

Por forma a facilitar a análise às BD locais, foi criado um executável[12] recorrendo ao NDK *Android* e ao código fonte do SQLite[13] em C. A compilação criada para *armv7* e o executável criado para ler conteúdo de bases dados em *SQLite*, após a sua criação foi exportado para o dispositivo de testes. Em seguida foram atribuídas as permissões de acesso ao mesmo, ficando assim disponível para uso por parte do analista a partir da *shell* do DM.

Desta forma é possível ler o conteúdo das DB presentes no dispositivo, sem ser necessário para isso copiar para a máquina local e em seguida analisar o seu conteúdo. Facilitando a análise principalmente se existir alterações frequentes, na qual seja necessário consultar os valores presentes na BD. Uma alternativa similar ao *sqlite3* é a ferramenta *Objection*, descrita na Secção 4.3.3 página 74 deste trabalho.

4.2.10 *Termux*

Foi instalado o *termux*, um emulador de terminal *open source* para *Linux*. A motivação para tal foi a necessidade de correr um servidor de Secure Shell (SSH) no DM. A porta por omissão de SSH no *termux* é a TCP/8022.

É possível aceder a uma *shell* com privilégio de `root` no DM, de uma forma segura (SSH) sem a necessidade de qualquer cabo para estabelecer a comunicação.

No *host*, foi necessário criar um par de chaves pública/privada. A chave pública foi posteriormente exportada para o DM de testes, através do comando presente na Listagem 3, mantendo a chave privada no dispositivo *host*.

Desta forma é possível estabelecer uma ligação através de um túnel SSH, com o exemplo disponível na Listagem 4 onde os comandos que são inseridos e os resultados obtidos são encriptados por um canal seguro de comunicação.

12 Sqlite3 Client Android, aquisição BD - Disponível em: https://github.com/stockrt/sqlite3-android - Acedido a 19 de Dez. 2022

13 SQLite amalgamation - Disponível em: https://www.sqlite.org/download.html - Acedido a 19 de Dez. 2022

Listagem 3: Exportar a chave pública para o DM

```
1    ssh-copy-id -p 8022 -i id_rsa 192.168.1.x
```

Listagem 4: Estabelecer a ligação SSH do host ao DM remoto

```
1    ssh -p 8022 192.168.1.x
```

4.2.11 *Droidlysis*

O *droidlysis* é uma ferramenta que faz o *disassemble* da aplicação com o intuito de procurar no código fonte por diferentes informações relevantes para tarefas de engenharia reversa da aplicação. O *droidlysis* utiliza ferramentas *open-source* para executar diferentes tarefas e por fim compilar toda a informação obtida num único relatório, que encontrado-se dividido pelas Listagens 48 (página 184), 49 (página 185), 50 (página 186), 51 (página 187), 52 (página 188). O *droidlysis* faz uso de ferramentas para a análise da aplicação, sendo as relevantes o *Apktool*, *Baksmali*[14], *Dex2jar*[15] e *Procyon*[16].

O relatório criado por esta ferramenta é um bom ponto de partida se o analista pretender ter uma visão geral dos recursos que são utilizados por uma aplicação, e encontra-se dividido da seguinte forma:

1. Características do APK;
2. Características do Certificado;
3. Características do *Manifest*;
4. Características dos recursos (`/res`);
5. Identificação de funcionalidades através da interpretação de código *Smali*;
6. Bibliotecas de terceiros e SDK's utilizados pela aplicação;
7. Conjunto de propriedades não classificadas (Uniform Resource Locator (URL)'s, *strings*, etc..).

14 Baksmali - Disponível em : https://bitbucket.org/JesusFreke/smali/downloads/ - Acedido a 24 Dez. 2022
15 Dex2Jar Git - Disponível em: https://github.com/pxb1988/dex2jar - Acedido a 24 Dez. 2022
16 Procyon Github - Disponível em: https://github.com/mstrobel/procyon/releases/tag/v0.6.0 - Acedido a 24 Dez. 2022

4.2.12 *Get Schemas*

A ferramenta *Get Schemas* foi utilizada para se obter todas as combinações possíveis de *deep links*. Esta também é uma ferramenta de código aberto[17], a qual permite obter a listagem de todos os URL *schemes* e as suas correspondentes *activities*.

Ambos os ficheiros `AndroidManifest.xml` e `strings.xml` foram obtidos durante as tarefas de engenharia reversa anteriormente realizadas. O ficheiro `strings.xml` está localizado em `/res/values/strings`, a localização do `Manifest` já foi abordada anteriormente.

Através do comando descrito na listagem 5, foi possível obter a lista dos *schemes*, *host* e *path* para cada uma das *activities* existente, na qual é feito o uso de *Deep Links*.

Listagem 5: Ferramenta: get_schema - App: Bolt

```
1 python3 get_schemas.py -m AndroidManifest.xml -s strings.xml
```

4.2.13 *Ferramentas de análise estática*

A tabela 8 da página 69, contém as ferramentas mais relevantes para análise estática que foram utilizadas no decorrer deste trabalho. Não foi feito um estudo comparativo de resultados obtidos em relação a outras ferramentas existentes no mercado, que desempenham papeis semelhantes. Os resultados obtidos pelas diversas ferramentas aqui descritas, estão sujeitas a alterações nos resultados finais que podem ser significativas, que podem ser consequência das versões utilizadas, bem como das aplicações móveis em análise.

17 https://github.com/teknogeek/get_schemas- acedido a 20 de julho de 2023

Tabela 8: Ferramentas de análise estáticas utilizadas

Ferramenta	**Utilização prática**
ADB	Utilizado para instalação de aplicações, transferência de arquivos e execução de comandos shell.
Apktool	Utilizado nas tarefas de *disassembly* e reconstrução do APK da aplicação de estudo, permitindo alterar o comportamento da aplicação.
Androguard	Permitiu criar gráficos de controlo de fluxo, para simplificar a leitura de classes com elevada obfuscação.
dex2jar	Conversão de ficheiros DEX(formato *Dalvik bytecode*), para o formato JAR, tornando o código mais acessível para análise estática. Facilitando a compreensão do código *Java* subjacente.
Logcat e pidcat	Permitiu filtrar *logs* da aplicação, facilitando o *debug* e análise de registos específicos.
Jd-Gui	Visualização do código fonte *Java* da aplicação *Bolt*.
Jadx	Facilitou todo o processo de análise estática, ao converter de forma simplificada o ficheiro APK em código-fonte Java legível, permitindo destacar e pesquisar por métodos, variáveis, *keywords*, etc..
Droidlysis	Efetuar verificações de segurança de uma forma abrangente e gerar relatórios detalhados, auxiliando na identificação de potenciais vulnerabilidades.
Get Schemas	Permite a análise dos URL *schemes*, ao associar o URL/URI à *activity* correspondente.
App Link Verification	Teste automático e validação de todos os *links* em uso pela aplicação.

4.3 FERRAMENTAS ANÁLISE DINÂMICA

As ferramentas mais relevantes que foram utilizadas durante a fase de análise dinâmica são seguidamente apresentadas de forma sucinta. O objetivo desta secção é compreender as vantagens decorrentes do seu uso e como estas podem ajudar a efetuar uma análise dinâmica.

4.3.1 *Uber apk signer*

Esta ferramenta facilita todo o processo de assinatura do APK, ao efetuar o *zipalign*[18] e a verificação do APK após introduzir alterações ao código fonte. Este tipo de ação pode ser necessário quando pretendemos alterar o comportamento de uma aplicação durante o processo de análise dinâmica, permitindo 'guardar' as alterações efetuadas durante a análise dinâmica e assim ganhar tempo durante a análise da aplicação. A ferramenta suporta o esquema de assinaturas Android v1, v2 e v3, utilizando a sua própria *keystore* com chaves de depuração que verifica de forma automática após cada assinatura. Esta é mais uma ferramenta de código aberto, a qual podemos encontrar na página do seu criador[19]. O projeto encontra-se ativo e em evolução à data deste trabalho.

4.3.2 *Frida*

O Frida é uma ferramenta *open source* criada por Ole André Vadla Ravnås e patrocinada pela empresa *NowSecure*, desenvolvida para instrumentalização de código de forma dinâmica, injetando *QuickJS* no processo a ser instrumentalizado. O *QuickJS*[20] é uma implementação do mecanismo *JavaScript* (interpreta e executa *JavaScript*), criado para ser eficiente em termos de desempenho e tamanho, projetado para ser utilizado em sistemas com recursos limitados.

Os SO e arquiteturas suportadas pelo *Frida* são as seguintes:

- **Windows** (x86, x64);

18 Funcionalidade e uso da ferramenta ZipAlign - Disponível em: https://developer.android.com/tools/zipalign?hl=pt-br - Acedido a 22 Mar. 2024
19 Uber-apk-signer. Código fonte - Disponível em: https://github.com/patrickfav/uber-apk-signer - Acedido a 24 Maio 2023
20 Página oficial do QuickJS - Disponível em: https://bellard.org/quickjs/ - Acedido a 20 Nov. 2023

- **Linux** (x86, x64, arm, arm64, arm64e);
- **MacOS** (x86, x64, Apple Silicon M1);
- **Android** (incluindo x86);
- **iOS** (arm64, arm64e, x64).

Para melhor perceber o funcionamento interno da ferramenta *Frida*, identificam-se de seguida as suas principais características:

- **frida-core:** Faz parte do funcionamento interno do *Frida*, possibilitando a instrumentação com *JavaScript*, entre outras funcionalidades (comunicação bidirecional, enumeração de processos, etc.).
- **frida-python:** Responsável pela interacção com o *frida-core*, utilizando *bindings* (bibliotecas que permitem o acesso à API *Frida*).
- **frida-agent:** É uma biblioteca do *Frida* (*frida-core*) que é injetada no nosso processo-alvo e realiza tarefas de baixo nível, tais como a criação de *hooks*, comunicação com o nosso código de instrumentação. É escrita em *JavaScript*.
- **frida-gum:** É uma biblioteca[21] de instrumentação e introspecção escrita em C, que é também parte do *frida-core*.

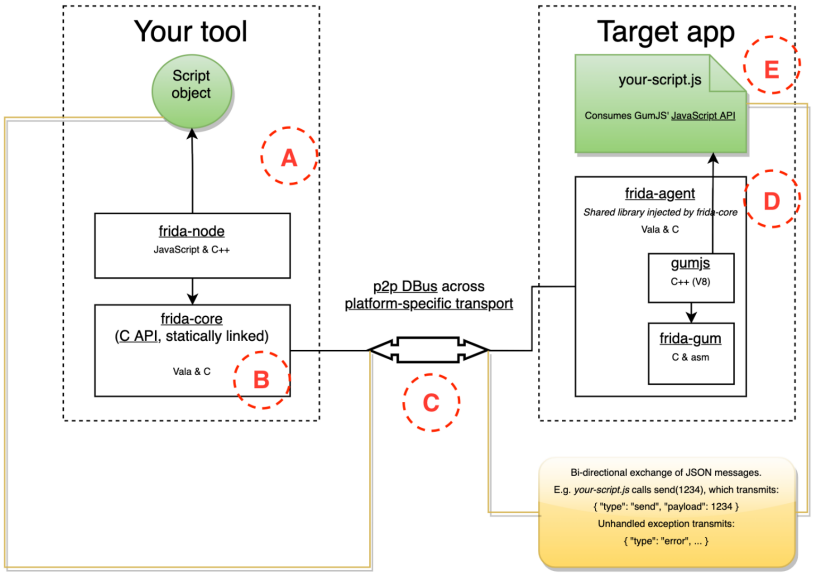


Figura 10: Arquitetura Frida

Fonte: https://frida.re/docs/hacking/

21 Frida-Gum, Disponível em: https://github.com/frida/frida-gum - Acedido a 22 Nov. 2023

Podemos ver do lado esquerdo da figura 10, assinalado com a letra **A**, a interface de controlo da ferramenta. No caso deste trabalho prático foi utilizada a Read evaluate print loop (REPL) *Frida*, a qual comunica através de *bindings* com o *frida-core* assinalado com a letra **B**. Do lado direito da figura 10 que se encontra assinalado com a letra **D**, encontra-se o *frida-agent*. Este é injetado no processo da aplicação em análise e que interage com o código em *Javascript* que é por nós criado, representado pela letra **E**. A comunicação entre o *frida-core* e o *frida-agent*, é efetuada através do **DBUS**, um sistema de comunicação entre processos (IPC), assinalado com a letra **C**. Este sistema possibilita a comunicação entre o *frida-agent* e o código *JavaScript* que está a ser executado, permitindo assim a troca bidirecional de mensagens no formato JavaScript Object Notation (JSON).

O *Frida* pode operar de três formas distintas[22], conforme a seguir indicado:

- ***Injected*:**
  - Instrumentação dinâmica de programas existentes em execução, que requer um dispositivo móvel *rooted*.
  - Utiliza o *frida-core* para injetar *GumJS* na aplicação a analisar.
  - *GumJS* é o componente que contém a infra-estrutura para instrumentalização dinâmica. Este componente é escrito em linguagem C.
  - O *Frida-core* estabelece um canal bidirecional de comunicação, para executar a lógica da instrumentalização.
  - O *Frida-server* é executado no dispositivo móvel (*Android/iOS*), é o componente que expõe o *frida-core* sobre Transmission Control Protocol (TCP).
- ***Embedded*:**
  - Modo de uso empregue quando não é possível utilizar um sistema *rooted* (*Android* ou *iOS*).
  - Utiliza o *frida-gadget*, que é uma biblioteca embebida dentro do programa a instrumentar.
  - Permite interação remota através do *frida-trace* (observar e seguir chamadas de funções, argumentos, etc.. em tempo real) e execução autónoma de *scripts* presentes no sistema de ficheiros.
- ***Preloaded*:**

22 Frida: Modos funcionamento - Disponível em: https://frida.re/docs/modes/ - Acedido a 17 Out. 2022

- Modo de uso quando não é possível utilizar um sistema *rooted* (*Android* ou *iOS*).
- Ao contrário do modo *embedded* que inclui a biblioteca diretamente no programa, o *preloaded* carrega o *frida-gadget* durante a inicialização do programa.
- Tal como o *embedded*, este modo permite interação remota através do *frida-trace* e execução autónoma de *scripts* presentes no sistema de ficheiros.

Neste trabalho foi empregue um DM *rooted*, tendo se feito uso do modo *injected*.

METODOLOGIA DE USO DO FRIDA Seguidamente, descreve-se a metodologia de uso do Frida que foi empregue na análise dinâmica. É dada especial atençao às dificuldades nas interações com o *Frida* e às opções tomadas para as ultrapassar.

É necessário que as versões do *Frida* que são utilizadas, tanto na máquina do analista (*frida-client*) como no dispositivo móvel (*frida-server*), tenham ambas a mesma versão. A arquitetura utilizada pelo dispositivo móvel para esta análise foi a *ARM64* e as ferramentas de cliente utilizadas na máquina de análise *x86_64*. Sendo esta uma aplicação *open-source*, temos acesso a todas as versões no respetivo *github*[23].

A primeira dificuldade encontrada, ocorreu no início, quando se selecionou a versão *Frida* 16.0.5. Esta era a versão mais atual à data de início deste trabalho e como tal foi o ponto de partida. Após exportar ambas as aplicações para os dois dispositivos e confirmar que se estava a fazer uso da mesma versão em ambos os dispositivos, foram executados comandos para correr no dispositivo móvel. Foram executados com sucesso a execução de simples comandos para listar os processos em execução no DM, verificando-se assim a operacionalidade da comunicação entre o dispositivo móvel e a máquina do analista.

No entanto, ao iniciar a instrumentalização da aplicação não foi possível lançar com a *flag -f*, a aplicação numa nova instância. A única opção seria pois o uso da *flag -F* (ex: *frida -U -F*), modo no qual é feito o *attatch* à aplicação em *foreground* (aplicação visível ao utilizador). Além disso qualquer posterior instrumentalização bloqueava a aplicação, o que obrigava a reiniciar o processo *frida-server*, o que devolvia a aplicação ao seu estado normal.

23 Releases Frida, Github - Disponível em: https://github.com/frida/frida/releases - Acedido a 5 nov. 2022

Ao ser confrontado com este problema e devido ao dispositivo de teste ter uma versão *Android* bastante obsoleta (*Android* 5.0.2 *Lollipop*), optou-se pela atualização para a versão *Android* 6.0 *Marshmallow* do DM, solução esta que se revelou infrutífera. De referir que a versão *Android* 6.0 para o DM em uso, foi a última versão *OEM* suportada pelos fabricantes do dispositivo, para o modelo utilizado.

Visto que o problema encontrado se manteve, optou-se por uma *custom* ROM intitulada de 'Pixel Experience'[24]. Esta utiliza *Android* 9.0 e foi criada para o DM em uso (*Xiaomi Redmi Note 3*). Sendo baseada numa *ROM* AOSP era a escolha que trazia uma maior garantia de compatibilidade para o *hardware* do dispositivo.

Após serem realizados novos testes e se constatar que o problema se mantinha, ficou demonstrado que a causa do problema encontrado não era a versão do *Android*.

Foram então realizados testes com diferentes versões *Frida*, sendo que foram utilizadas várias versões que foram lançadas ao longo dos anos. Partindo da iteração mais recente foram testadas as versões por ordem decrescente, garantindo sempre que ambas as versões, tanto no DM como na máquina do analista, eram as mesmas. Assim se chegou à versão *Frida* 15.2.2. Esta versão mostrou-se totalmente funcional, apresentando o comportamento esperado e sem qualquer tipo de falha na execução dos comandos introduzidos durante a instrumentalização da aplicação.

Importa notar que é necessário instalar as ferramentas na máquina do analista para ser possível comunicar com o *frida-server* presente no DM e como tal, a versão correspondente é denominada por *frida-tools*. A versão utilizada foi a *frida-tools 11.0.0*.

A versão com as ferramentas (*frida-tools*) necessárias, foi obtida através do comando 'pip install frida-tools==11.0.0'. Esta versão de ferramentas a instalar na máquina do analista correspondente à versão 15.2.2 *Frida* que se encontra no dispositivo móvel.

4.3.3 *Objection*

O *Objection* é uma ferramenta que permite auxiliar a realização de testes de segurança a aplicações móveis, criada pela empresa *SensePost*[25]. Este *software*

24 Pixel Experience ROM - Disponível em: https://forum.xda-developers.com/t/rom-9-0-kenzo-pixel-experience-aosp-r16-official-2018-11-10.3835648/ - Acedido a 22 fev. 2022

25 Objection Git - Disponível em: https://github.com/sensepost/objection - Acedido a 21 dez. 2022

permite-nos identificar e explorar vulnerabilidades em aplicações móveis, bem como obter diversa informação do dispositivo móvel.

O *Objection* recorre à *framework open-source Frida*, a qual permite injetar código *JavaScript* em aplicações e processos em execução no DM, conforme anteriormente visto.

Algumas das funcionalidades mais relevantes são:

- *Hooking* e modificação do comportamento de funções da aplicação;
- *Dump* de memória de processos em execução;
- Extrair dados do dispositivo;
- Ultrapassar controlos de segurança do DM, tais como *SSL pinning* e deteção de *root*.

Importa salientar que a versão *Frida* deve ser indicada previamente no ambiente virtual, de forma a corresponder à instalada no DM. Se tal não for feito, o *Objection* irá utilizar sempre a última versão disponível no *github Frida*.

Uma vez que parte do código da aplicação em estudo se encontra ofuscado, o uso do *Objection* é útil para auxiliar na instrumentalização da aplicação.

Um dos exemplos que podemos destacar é o uso da instrução indicada na Listagem 6.

Listagem 6: Obter a Activity em Foreground

```
1    ee.mtakso.client on (Xiaomi: 9) [usb] # android hooking get current_activity
2    Activity: ee.mtakso.client.newbase.RideHailingMapActivity
3    Fragment: lv.f
```

Devido à sua grande dimensão, a informação resultante do comando acima indicado encontra-se disponível no apêndice B.1, página 193 e serve como referência ao longo deste documento na realização de testes à aplicação.

4.3.4 *radare2*

O *Radare2* também conhecido por R2[26] é um conjunto de ferramentas para engenharia reversa de aplicações móveis. Este pode ser utilizado para tarefas forenses sobre os ficheiros do sistema independentemente do SO, ou em contexto de testes de

26 Página oficial, Radare2 - Disponível em: https://www.radare.org/n/radare2.html - Acedido a 12 Mar. de 2024

penetração e/ou segurança, para tarefas de *disassembly*, análise, emulação e *debug* de aplicações.

A curva de aprendizagem do uso desta ferramenta é muito acentuada e o mesmo é referido pelos criadores da ferramenta. As principais limitações do uso da mesma devem-se à capacidade limitada de extrair conteúdo relevante resultante do processo de análise.

Tal como as outras ferramentas utilizadas neste trabalho, também esta é *open-source* e encontra-se em constante evolução. Foi inicialmente criada por `Sergi Àlvarez` em 2006, um analista forense e o seu intuito era desenvolver um editor hexadecimal que lhe permitisse recuperar ficheiros apagados de uma partição Mac OS Extended (HFS)[27].

4.3.5 *R2Frida*

O *R2frida* é um *plugin* para o *Radare2* que combina as capacidades de análise estática do *Radare2* com as capacidades de *hooking* e *tracing* dinâmico do *Frida*. O *R2frida* destaca-se pela capacidade de realizar *tracing* ao comportamento da aplicação, e na manipulação de código em execução, sendo estes os pontos fortes que a tornam numa ferramenta importante de engenharia reversa e segurança.

O *R2frida* lê e escreve na memória através do *Frida*, e utiliza o *Radare2* para o *disassembly* e análise estática, possibilitando o uso de comandos específicos do *Frida* para definir *hooks* ou realizar *tracing* durante a execução de código. O *R2frida* é implementado como um *plugin* de Input/Output (IO) para o *Radare2*, utilizando a mesma forma de leitura e escrita em memória à empregue pelo *Radare2* numa análise à memória local.

O *R2frida* permite trabalhar com aplicações locais, mas o propósito da sua criação é a instrumentalização de aplicações móveis em execução num dispositivo físico *Android* ou *iOS*.

A figura 11 ilustra a interação do *Radare2* com o *Frida*.

27 Livro Radare2 - Disponível em: https://book.rada.re/ - Acedido a 16 Ago. 2023

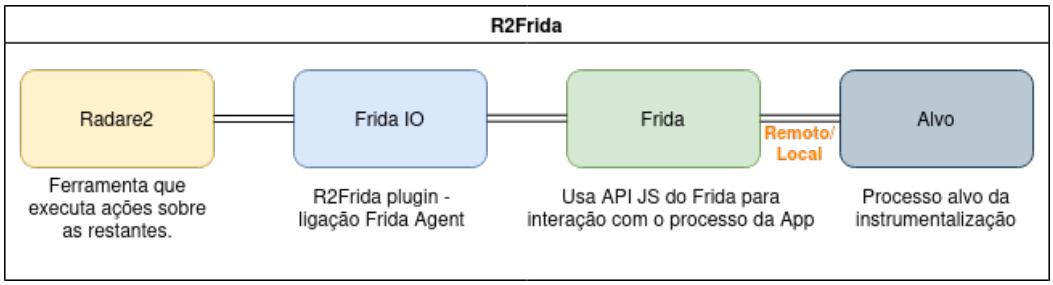


Figura 11: R2Frida - Representação dos seus componentes

Fonte: https://github.com/radareorg/r2con2021/blob/master/day2/r2frida_2021-r2con.pdf

4.3.6 *Mitmproxy*

Por forma a analisar o tráfego de rede da aplicação em tempo real com sucesso, é necessário que a informação contida em cada pacote seja legível. Tendo a perceção que a informação transmitida entre dispositivos é na sua grande parte encriptada é necessário que sejam usadas ferramentas que nos permitam desencriptar a informação que chega ao dispositivo.

O *Mitmproxy*[28] disponibiliza um conjunto de recursos que nos permitem de forma interativa, intercetar comunicações, sejam estas SSL/TLS para Hypertext Transfer Protocol (HTTP)/1, HTTP/2 e a nova adição HTTP/3. Apesar de já existir a possibilidade de uso para HTTP/3, este apenas pode ser utilizado por uma *proxy* reversa[29], como seria de esperar é também feito o reencaminhamento de tráfego Quick UDP Internet Connections (QUIC).

Existem ainda outras ferramentas para realizar esta mesma tarefa, tais como *Fiddler*[30], BurpSuite, OWASP ZAP, Lemur[31], Charles[32] e HTTP *Toolkit*. Para além dessas ferramentas de código aberto, existem outras opções proprietárias que não foram consideradas para a execução deste trabalho. Cada uma das ferramentas apresenta pontos fortes e fracos, de acordo com a distribuição de SO para o cenário de testes, objetivos e funcionalidades pretendidas, bem como o custo da solução.

28 Mitmproxy - Disponível em: https://docs.mitmproxy.org - Acedido a 25 Mar. 2024
29 MiTM suporte Http e Quick - Disponível em: https://mitmproxy.org/posts/releases/mitmproxy10/ - Acedido a 25 Mar. de 2024
30 Fiddler - Disponível em: https://www.telerik.com/fiddler - Acedido a 09 set. 2022
31 Lemur - Disponível em:https://github.com/Netflix/lemur - Acedido a 09 set. 2022
32 Charles - Disponível em: https://www.charlesproxy.com/ - Acedido a 09 set. 2022

4.3.7 *TCPDump*

O TCPDump é uma ferramenta de *sniffing* empregue com o intuito de se obter, em tempo real, todo o tráfego de rede que chega ao DM. Neste trabalho, o TCPDump foi utilizado para a recolha do tráfego para um ficheiro '.pcap', sendo esses ficheiros posteriormente analisados com a aplicação 'Wireshark' (4.3.8).

Em primeiro lugar foi necessário obter a aplicação *tcpdump* compilada para dispositivos com a arquitetura Advanced RISC Machine (ARM)[33] e em seguida passar o mesmo para o DM.

Listagem 7: Dispositivo Móvel: Transferência, configuração e definição de permissões do TCPDump

```
1  $ adb push tcpdump /data/local/tmp
2  $ adb shell
3  $ su
4  $ mount -o rw,remount /system
5  $ cp /data/local/tmp/tcpdump /system/xbin
6  $ cd /system/xbin
7  $ chmod 755 tcpdump
```

Para o efeito, foi empregue a abordagem apresentada na Listagem 7, dado que não era possível montar o diretório como `root` (mesmo tendo permissões de `root`).

A linha de comandos para a execução do *TCPDump* no dispositivo móvel é mostrada na Listagem 8.

Listagem 8: Dispositivo Móvel: Enviar tráfego de rede recolhido pelo TCPDump através do Netcat

```
1  $ ./tcpdump -i wlan0 -s0 -w - | toybox nc -l -p 5555
```

Essa linha de comandos realiza as seguintes tarefas:

1. Escuta o tráfego recebido na interface `wlan0`;
2. Define o tamanho da captura em bytes através da *flag* -s0 (obtém todo o tráfego);
3. Escreve o conteúdo para um ficheiro (-w), mas em vez de um nome do ficheiro passamos -, o qual irá enviar o resultado para o `stdout`.
4. Usa o `toybox` para correr o comando `netcat`, não sendo assim necessário instalar esta aplicação à parte.

33 Android tcpdump, Download - Disponível em: https://www.androidtcpdump.com/android-tcpdump/downloads - Acedido a 10 Jun. 2023

5. O conteúdo do `tcpdump` é enviado via 'pipe' (|) para o `netcat`, este último disponibiliza o conteúdo recolhida através da porta 5555.

Note-se que para ser possível aceder à porta 5555 é preciso inserir o comando indicado na Listagem 9.

Listagem 9: Máquina analista: Redirecionar tráfego Android da porta 5555 para a porta 5555 da máquina do analista

```
$ adb forward tcp:5555 tcp:5555
```

No terminal da máquina do analista, é necessário identificar a porta previamente estabelecida (5555), bem como a aplicação Wireshark que lê e processa o tráfego recebido. Tal é feito com a linha de comandos indicada na Listagem 10.

Listagem 10: Máquina analista: Alimentar o Wireshark com conteúdo recebido pelo netcat

```
$ nc localhost 5555 | wireshark -k -S -i -
```

A figura 12 sintetiza os pontos mais importantes do *setup*, visando dar ao leitor uma melhor prespetiva dos comandos a serem introduzidos nos diferentes equipamentos.

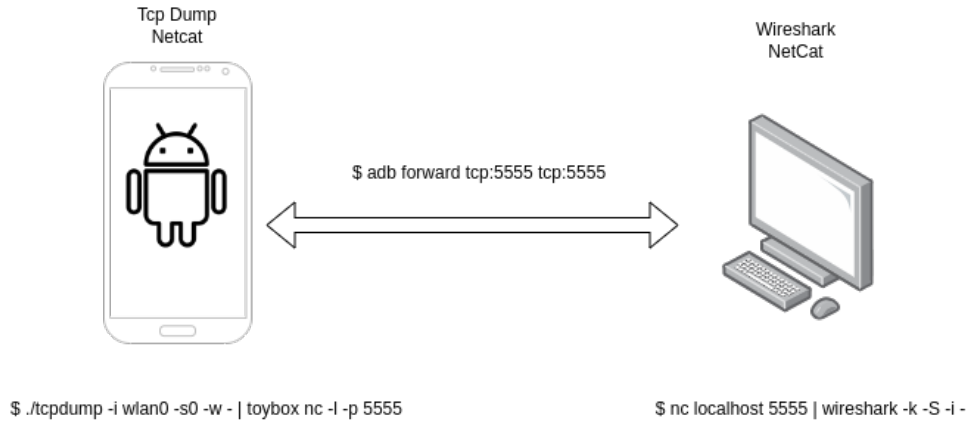


Figura 12: Resumo setup TcpDump

A figura 13 ilustra o resultado final da captura e leitura no `Wireshark` do tráfego capturado do DM. Esta ferramenta por suportar a análise dos protocolos utilizados pela aplicação e perceber se o mesmo é encriptado ou não. Além disso, este *setup* permite identificar quais são os *endpoints* que a aplicação utiliza para comunicar.

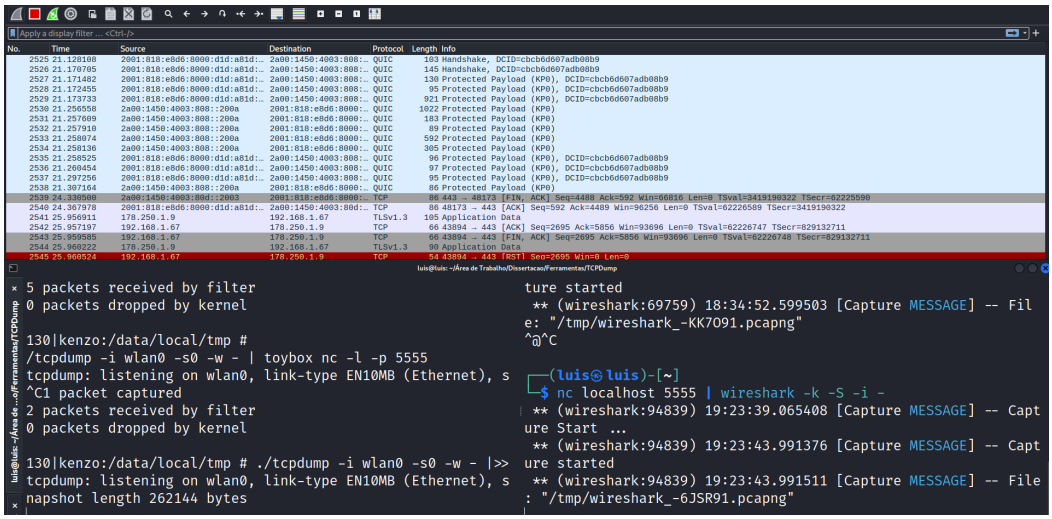


Figura 13: Resultado do setup para recolha de tráfego de rede

4.3.8 *Wireshark*

A ferramenta Wireshark[34] é utilizada por diversos profissionais na àrea de segurança para, entre outras tarefas, a análise de protocolos de rede. Neste trabalho foi utilizada para ler o tráfego obtido pelo TCPDump, recolhido diretamente do DM, conforme anteriormente indicado 4.3.7.

Não foi necessário nenhuma configuração adicional além da sua instalação para procedermos ao seu uso, sendo este também bastante intuitivo. Ao longo deste trabalho serão ainda destacadas algumas funcionalidades da ferramenta, nomeadamente a filtragem do conteúdo a ser analisado.

4.3.9 *BurpSuite*

O *BurpSuite*, ou simplesmente *Burp*, representa um conjunto de ferramentas que auxiliam na execução de testes de segurança para aplicações *web*. É desenvolvido pela empresa *Portswigger*, cujo fundador é Dafydd Stuttard, tendo a primeira versão do *BurpSuite* sido lançada em Junho de 2003.

O *BurpSuite* é utilizado neste trabalho para servir como *proxy* de intercepção, que nos permita interceptar, inspecionar e modificar os pedidos e respostas que são trocados entre a aplicação *Bolt* presente no DM e os serviços remotos de forma a testar o comportamento e eventuais vulnerabilidades existentes.

34 Wireshark - Disponível em: https://www.wireshark.org/ - acedido a 5 de Jun. 2023

O *BurpSuite* dispõem de três soluções para realizar a tarefa anteriormente descrita, que são:

- **BurpSuite Enterprise Edition:** Scanner de vulnerabilidades web dinâmico;
- **BurpSuite Professional:** *Toolkit* de ferramentas para testes de penetração web;
- **BurpSuite Community Edition:** Ferramentas não automatizadas (básicas) para testes de segurança a aplicações web;

A solução utilizada para a criação deste trabalho foi a *BurpSuite Community Edition* '**v2023.10.3.5**'. A principal razão que levou à escolha desta opção (*Community Edition*) foi esta ser a única solução gratuita em relação às restantes opções. A solução *BurpSuite Professional*, que seria a mais indicada para a realização dos testes pretendidos requer uma subscrição de 450€/ano.

Algumas das vantagens do *BurpSuite* em relação a outras ferramentas disponíveis é a capacidade de juntar um conjunto de funcionalidades que são consideradas como básicas e indispensáveis numa única plataforma para testes de segurança. Acresce-se ainda queas capacidades podem ser aumentadas com a instalação de complementos designados por *BApps*.

As ferramentas mais relevantes que fazem parte do *BurpSuite* são as seguintes:

- **Spider:** Permite a recolha e identificação de páginas e funcionalidades que podem ser acedidas, útil para a descoberta de elementos ocultos.
- **Scanner:** Realiza o *scan* automático de vulnerabilidades, não funciona na versão *Community*. Devido a este fator, não foi possível a sua utilização.
- **Intruder:** Automatizar testes de parâmetros de entrada, para identificar vulnerabilidades como injeção de SQL e Cross-site Scripting (XSS).
- **Repeater:** Possibilita a manipulação, de forma manual, do reenvio de pedidos HTTP individuais. Este é considerada uma das funcionalidades essenciais no processo de teste para aplicações web, tendo sido amplamente empregue neste trabalho.
- **Sequencer:** Verifica se há algum padrão explorável ou falta de aleatoriedade nos *tokens* ou valores de sessão, o que poderia ser explorado por potenciais atacantes.
- **Decoder:** Descodifica/codifica dados em diferentes formatos, suportando codificação/descodificação de URL, *octal*, *base64*, *binary*, entre outros.

A vantagem do *BurpSuite* é utilizar as funcionalidades anteriormente descritas de forma combinada. Um exemplo prático é enviar o resultado obtido do *Intruder* para o *Repeater* e testar de forma manual o resultado obtido. O *BurpSuite* também pode ser empregue de maneira cíclica: Usando-se o *Decoder* para decodificar os dados das respostas ou pedidos e realizar transformações com o *Repeater*, é possível modificar e testar esses novos valores novamente no *Repeater* para observar os resultados obtidos.

A figura 14 ilustra a rede empregue no ambiente de teste, bom como a localização do *BurpSuite*.

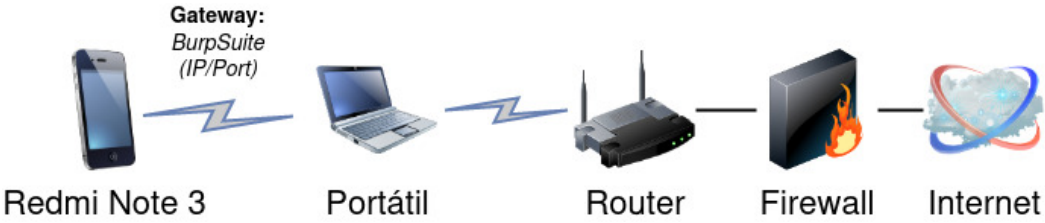


Figura 14: Esquema de rede - Burpsuite

UTILIZAÇÃO DO BURPSUITE. Inicialmente começamos por obter o endereço Internet Protocol (IP) da máquina de testes, sendo que foi utilizada uma interface de rede dedicada para as tarefas de análise. A razão da mesma é para que não existam pedidos de outras aplicações que estejam a correr na máquina do analista que interfiram na análise da aplicação. O uso de uma interface dedicada também facilita a recolha de tráfego através de ferramentas como o *Wireshark*.

É também importante definir um porto que não esteja a ser utilizado por outra aplicação, para facilitar a filtragem do tráfego a analisar. O processo anteriormente descrito pode ser visto através da Figura 15.

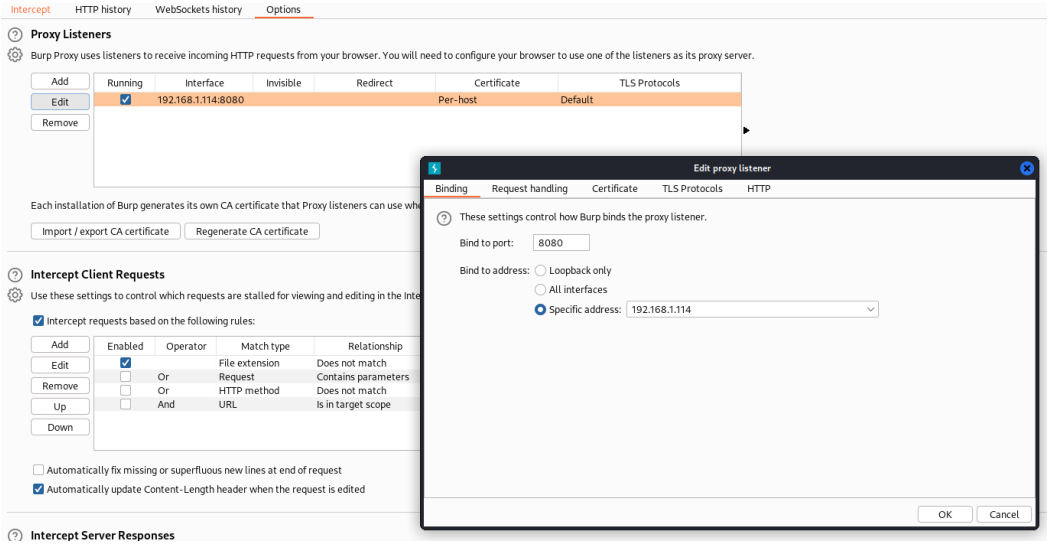


Figura 15: Definir IP/Porto do Gateway no BurpSuite

4.3.10 *Ferramentas de análise dinâmica*

A Tabela 9 lista as ferramentas mais relevantes de análise dinâmica empregues neste trabalho. Tal como foi referido na secção 4.2.13, a escolha das ferramentas apresentadas neste trabalho representam apenas uma pequena parte das existentes no mercado. Os fatores de escolha mais determinantes foram a existência de documentação e suporte que ofereça integridade dos resultados obtidos, serem gratuitas/código aberto e que sejam reconhecidas por profissionais da área de segurança quanto à qualidade dos resultados produzidos.

Tabela 9: Ferramentas de análise dinâmica utilizadas

Ferramenta	**Utilização prática**
Uber apk signer	Auxiliar o processo de verificação e assinatura digital do APK.
Frida	Criação de *scripts* que permitiram a manipulação dinâmica da aplicação *Bolt*.
Objection	Permitiu desempenhar diversos propósitos: *Dump* das BD locais e da memória dos processos em execução, desativar SSL *pinning*, *hooking* de funções, etc..
radare2/ R2Frida	Ferramenta e *plugin* para análise do conteúdo em RAM da aplicação móvel.
TCPDump	Recolha do tráfego de rede do dispositivo móvel, gerado pela aplicação *Bolt*.
Wireshark	Leitura e análise dos pacotes de rede recolhidos pelo *TCPDump*.
MitmProxy	Interceção e leitura dos pedidos/respostas da aplicação.
Burp Suite	Interceção e manipulação de pedidos/respostas web, com o objetivo de testar diversos fatores de segurança e vulnerabilidade da aplicação *Bolt*

Este capítulo apresentou a lista de ferramentas para análise estática e análise dinâmica de aplicações móveis. No capítulo seguinte procede-se à análise estática da aplicação *Bolt*.

5

ANÁLISE ESTÁTICA

Este capítulo explora os principais aspetos e metodologias para a realização de uma análise estática. Após a análise da política de privaciade da *Bolt*, é apresentado o processo de descompilação de uma aplicação móvel. As restantes secções deste capítulo são dedicadas à análise dos ficheiros resultantes da descompilação.

5.1 POLÍTICA DE PRIVACIDADE

Uma 'Política de Privacidade' é uma explicação detalhada dos dados pessoais recolhidos, tratados e guardados pelo criador da aplicação, incluindo o motivo da sua recolha. Este tipo de documentos servem como documentos legais empregues para indicar deveres e direitos dos clientes e da empresa.

As informações contidas numa 'Política de Privacidade' são de extrema relevância. Embora possa aparentemente facilitar o trabalho do analista, este último deve confrontar o definido na Política de Privacidade com o comportamento da aplicação.

Destacamos de seguida a informação mais relevante relacionada com a definição dos dados pessoais recolhidos pela aplicação, referentes ao seu utilizador e o período de retenção de dados. A informação que se segue, foi retirada da página *web Bolt*[1], referente à sua 'Política de Privacidade'. Para uma versão mais detalhada da Política de Privacidade recomendamos a leitura da mesma na sua totalidade.

Dados pessoais processados:

- **Detalhes de contacto:** Nome, número de telefone, endereço de email, e opcionalmente, morada de casa;
- **Informação de perfil:** Imagem de perfil, endereços guardados, língua, preferências de comunicação;
- **Geolocalização:** Local de partida, localização de trotinetas perto de si, hora, progresso da viagem e o destino final da viagem;

1 Política de Privacidade para Passageiros e Condutores, Bolt - Disponível em: https://bolt.eu/pt-pt/legal/pt/privacy-for-riders - Acedido a 12 Out. 2022

- **Informações de pagamento:** Montante cobrado, cartão de pagamento utilizado;
- **Registos de comunicação trocada:** Troca de mensagens em *chat* de bordo, ou entre agentes de apoio ao cliente;
- **Dados de identificação dispositivo:** Endereço IP;
- **Utilização de transporte:** Estado da viagem, horários, avaliação dos condutores;
- **Comunicação com o condutor:** Registo de mensagens instantâneas trocadas, data e hora da comunicação e seu conteúdo.

É também referido que "Quaisquer dados pessoais recolhidos no decurso da prestação dos nossos serviços são transferidos e armazenados nos centros de dados da *Zone Media Ltd.* e/ou *Amazon Web Services, Inc.*, que estão localizados nos territórios dos Estados Membros da União Europeia"[2].

No mesmo documento é referido que todos os dados armazenados são utilizados para "fins de investigação e fins científicos", e que os dados de geolocalização são anonimizados para não ser possível identificar um utilizador a partir dos mesmos.

Retenção de dados

- **Dados pessoais:** Os dados pessoais são mantidos enquanto a conta se encontrar ativa. A remoção ocorre somente após a eliminação da conta, excepto caso sejam necessários para cumprir com obrigações legais, fins contabilísticos, resolução de disputas ou prevenção de fraude;
- **Suspeita de infração criminal:** Atividade fraudulenta ou informação falsa, durante um período máximo de 10 anos;
- **Dados financeiros:** Durante três anos a contar da data da última viagem;
- **Disputas de pagamento:** Serão retidos até à data da sua resolução ou expiração de reclamações;
- **Histórico de viagens e utilização de serviços de transporte:** Durante três anos, após os quais serão anonimizados;
- **Desinstalação da aplicação:** Não provoca a eliminação dos dados pessoais, sendo estes ainda retidos durante um período de três anos. Após os quais se o cliente não pretender manter a sua conta ativa, a mesma será encerrada e os dados pessoais apagados.

2 Política de Privacidade para Passageiros e Condutores, Bolt - Disponível em: https://bolt.eu/pt-pt/legal/pt/privacy-for-riders - Acedido a 12 Fev. 2024

5.2 OBTER APK

Obteve-se o APK da aplicação através da recolha direta pelo DM físico, com o comando presente na Listagem 11. Desta forma garantiu-se que se está a usar a mesma versão da aplicação instalada no dispositivo.

Listagem 11: Android 9.0: Extrair APK do dispositivo

```
$ adb pull /data/app/ee.mtakso.client-a2XGAVyASjgXv-H_tx2sqg==/base.apk=ee.mtaks
↪  o.client/base.apk)
```

O comando utilizado para obter a *path*, encontra-se na Listagem 12.

Listagem 12: Listar path Android - package Bolt

```
$ adb -s 424ac404 shell pm list packages -f | grep -i "ee.mtakso.client"
```

Exemplos das diferentes *path*, onde podemos encontrar o APK responsável pela versão presente no DM da aplicação *Bolt* estão ilustradas na Tabela 10.

Tabela 10: Diferentes APK path de acordo com a versão Android - Bolt

Versão	**Path**
Android 6.0	`/data/app/ee.mtakso.client-1/base.apk=ee.mtakso.client`
Android 9.0	`/data/app/ee.mtakso.client-a2XGAVyASjgXv-H_tx2sqg==/ba se.apk=ee.mtakso.client`
Android 12.0	`/data/app/~~hUfSUwA76hJ8jqHJ1km3bQ==/ee.mtakso.client-k NpnSsu7_5qL3jKjZv_zHA==/base.apk=ee.mtakso.client`

Note-se que para versões posteriores a *Android* 8, o diretório `<appdir>`, de cada aplicação instalada contém um sufixo o qual acrescenta caracteres em formato *Base64* de forma a codificar a *path*. Por sua vez no *Android* 11 existe um sufixo extra também este codificado em *Base64*. A codificação da *Path* das aplicações foi criada com o intuito de prevenir outras aplicações (maliciosas) de obterem facilmente a lista de aplicações instaladas no dispositivo (Cybersecurity e IPLeiria, 2022).

5.3 DESCOMPILAR APLICAÇÃO

Para dar início à análise estática da aplicação, é necessário descompilar o APK de modo a ler o seu conteúdo. As aplicações em *Android* são distribuídas e instaladas no formato de ficheiro APK.

Listagem 13: Apktool: compilar/recompilar a aplicação

```
1  /* Descompilar a App */
2  $ apktool d Bolt52.1.apk
3
4  /* Recompilar a App */
5  $ apktool b Bolt52.1
```

O processo de descompilação/recompilação é bastante simples, conforme ilustrado na Listagem 13, que exemplifica as duas utilizações da ferramenta Apktool.

A descompilação permite-nos ter acesso ao ficheiro `AndroidManifest.xml`, no seu formato legível, o que é de extrema importância numa primeira abordagem.

Descrição do processo de descompilação

Ao descompilar a aplicação com recurso à ferramenta *Apktool*, esta não efetua a conversão dos ficheiros `classe.dex` para a sua versão Java, mantendo todos os ficheiros no formato .smali. Estes ficheiros são como linguagem *Assembly* para o SO *Android*, conforme observado na Secção 3.10.1.

Apesar de parecer que estamos a complicar o processo de engenharia reversa ao interagir com '.smali', estes passos são necessários quando se pretende modificar o código da aplicação e o analista não tem acesso ao código-fonte original da aplicação. As principais motivações para a modificação direta do código smali são as seguintes:

- Alterar o comportamento da aplicação;
- Melhorar a quantidade/qualidade de informação obtida;
- Injetar código para se perceber como a aplicação funciona;
- Contornar mecanismos de *anti-debug* da aplicação.

Neste documento não é detalhada a forma de escrever um programa em *Smali*, no entanto existem alguns cuidados que devem ser tidos em conta, ao editar/criar código *Smali* de forma direta. Ao programar em *Smali*, não são empregues os mesmos padrões de criação de código que um compilador tradicional utilizaria. Se as nossas alterações não forem reconhecidas durante o processo de descompilação, alguns dos métodos criados, podem simplesmente desaparecer do código descompilado. Esta

situação pode ser/'ou não' criada de forma intencional, o que leva à ofuscação do funcionamento da aplicação. Alguns dos exemplos que podem levar à necessidade de eliminar métodos podem-ser para a remoção do certificado x509 utilizado para o *certificate pinning*, ou para adicionar código que carregue bibliotecas, tais como Frida no ponto de entrada da aplicação.

Processo de obtenção de código Java

Se o propósito for apenas a leitura do código Java, existem ferramentas que nos auxiliam nessa tarefa. Destaca-se o *Jadx* referido na Secção 4.2.8, a qual foi utilizada neste trabalho. No entanto caso se pretenda guardar de forma permanente num formato *.jar*, as modificações de código, é necessário converter os ficheiros *.dex* para *.jar*.

Após o processo de extração do APK, temos acesso aos ficheiros *classes.dex*, *classes2.dex*, perfazendo 8 ficheiros com a extensão *.dex*, até ao ficheiro *classes8.dex*. Isto acontece porque nos arquivos APK, o número de métodos não pode exceder 65.536 métodos ou 64K por cada ficheiro *classes.dex*. Para ultrapassar essa limitação, desde o *Android* 5.0 é possível configurar o APK para suportar mais que um ficheiro DEX com a biblioteca '*multidex*'[3].

Para realizarmos a conversão, usa-se a ferramenta *dex2jar* 4.2.5, que tal como o nome revela, converte os ficheiros *".dex"* em ficheiros JAR. Posteriormente, com um editor de texto, ou um Integrated development environment (IDE), pode-se ler o código Java da aplicação, a figura 16 ilustra a execução do processo.

```
┌──(luis㉿luis)-[~/…/Dissertacao/Testes/ReverseEngineering/bolt52.1]
└─$ dex2jar *.dex
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
dex2jar classes2.dex → ./classes2-dex2jar.jar
dex2jar classes3.dex → ./classes3-dex2jar.jar
dex2jar classes4.dex → ./classes4-dex2jar.jar
dex2jar classes5.dex → ./classes5-dex2jar.jar
dex2jar classes6.dex → ./classes6-dex2jar.jar
dex2jar classes7.dex → ./classes7-dex2jar.jar
dex2jar classes8.dex → ./classes8-dex2jar.jar
dex2jar classes.dex → ./classes-dex2jar.jar
```

Figura 16: Converter os ficheiros .Dex para .Jar

Para leitura dos ficheiros JAR resultantes, optou-se pelo uso da ferramenta *JD-Gui* (4.2.7), que serviu como leitor do código fonte obtido. Esta ferramenta destina-se a tarefas de engenharia reversa, não sendo empregue para edição/compilação de

3 Multidex, Android Developers - Disponível em:https://developer.android.com/build/multidex - Acedido a 29 Nov. de 2023

código. A mesma ferramenta foi empregue para salvaguardar as classes Java em memória, e assim se ter finalmente acesso ao código no formato *.jar*.

Os ficheiros *.jar* podem ser importados para o IDE pretendido e a partir daí efetuar a manipulação do código-fonte, executando as modificações pretendidas. Este tipo de tarefa é principalmente importante quando estamos a trabalhar com código fonte que se encontra obfuscado, no qual pretendemos realizar alterações, tais como o renomear de determinados métodos ou classes, nomeadamente os mais relevantes. Este tipo de modificação auxilia o processo de leitura e análise, em aplicações complexas, possibilitando a identificação dos métodos que pretendemos manipular, no caso de uma análise dinâmica à aplicação.

5.4 ASSINATURAS E CERTIFICADOS

Para serem analisadas as assinaturas e certificados em uso pela aplicação *Bolt*, vamos apresentar o processo que foi seguido, destacando os pontos mais relevantes que devem ser tidos em conta durante uma análise estática.

Após a descompilação da aplicação, o diretório `META-INF` contém toda a informação respeitante aos certificados digitais da aplicação. Este diretório contém ainda informação relativa à versão de cada biblioteca e dependências necessárias para a execução da aplicação. Esta medida de segurança do *Android* visa garantir que apenas são utilizadas versões previamente testadas, que não alterem ou introduzam um comportamento divergente durante a execução da aplicação.

A nomenclatura aqui encontrada é gerada pela ferramenta `Apktool` de forma genérica ao descompilar o APK. Deve ser no entanto prestada atenção à extensão dos ficheiros de forma a conseguir identificar a sua função no processo de segurança da aplicação. Os ficheiros mais relevantes utilizados para o processo de assinatura de um APK, têm as seguintes extensões.

- **Cert.RSA / BNDLTOOL.RSA** : Ficheiro de assinatura;
- **Manifest.MF** : *Hash SHA-256* de cada ficheiro presente no APK.
- **Cert.SF / BNDLTOOL.SF** : *Hash SHA-256* da informação presente em `Manifest.MF` para cada uma das entradas.

Na aplicação em teste encontrámos o ficheiro `BNDLTOOL.RSA`, renomeado de acordo com o Apktool, ao qual corresponde ao ficheiro `Cert.RSA` na lista anterior. Este

ficheiro é guardado em formato binário e implementa uma mensagem com os seus valores criptográficos por omissão. Este ficheiro contém o valor da assinatura e o certificado assinado.

Ao exportar um APK assinado, este é assinado com a chave privada do certificado original. A chave pública é guardada dentro ficheiro `Cert.RSA`, em conjunto com a assinatura do ficheiro `Cert.SF` garantir a autenticidade e integridade do APK. A assinatura do ficheiro `Cert.SF` é um valor calculado por *hash*, este valor é encriptado com a chave privada e desta forma os resultados são guardados no ficheiro `Cert.RSA`.

Ambos os ficheiros `Manifest.MF` e `BNDLTOOL.SF` contêm resumos digitais no formato *SHA-256 digest*. No entanto o resultado (*output*) de ambos é distinto uma vez que o *input* utilizado para gerar a *hash* é também este distinto.

O propósito do ficheiro `Manifest.MF` é garantir a integridade de todos os ficheiros que constituem o APK. O *SHA-256* é aplicado sobre a totalidade do ficheiro que é utilizado como *input*.

O ficheiro `BNDLTOOL.SF` também conhecido como ficheiro de assinatura, (daí a extensão .SF de *signature file*), utiliza a informação de *metadata* contida no ficheiro *Manifest.MF* (nome + *hash*) para gerar uma nova *hash* a partir dessa informação. Além disso, o cabeçalho do ficheiro descreve as assinaturas utilizadas pela aplicação. No caso da aplicação *Bolt* existem assinaturas do tipo V1 e "*X-Android-APK-Signed*: 2, 3". Indicando que a aplicação foi assinada utilizando as versões de assinatura V1, V2 e V3 do esquema de assinaturas do APK. Essas versões de assinaturas são utilizadas para garantir a integridade e autenticidade da aplicação durante o seu processo de distribuição. O uso dessas versões também inclui proteções contra ataques, por exemplo, evitando que um APK assinado com o esquema V2 seja erroneamente verificado como se fosse assinado pelo esquema V1, especialmente em plataformas que suportam a verificação de APK no formato V2. Esse tipo de ataque é designado por ataque de regressão e tal como o nome indica traduz-se numa tentativa de reduzir a assinatura utilizada, ao alterar a assinatura V2 para uma V1. Essa proteção é implementada através da inclusão do atributo *X-Android-APK-Signed* nos ficheiros `META-INF/*.SF`, especificando as versões de esquema de assinatura usadas[4].

Dessa forma, a finalidade de ambos os ficheiros –`BNDLTOOL.SF, Manifest.MF`– é garantir a integridade de todos os ficheiros presentes no APK, funcionando como mecanismo de deteção de "*tampering*"e de corrupção de todos os ficheiros do APK.

4 APK Signature Scheme, Android Source - Disponível em:https://source.android.com/docs/security/features/apksigning/v2 - Acedido a 1 de Dez. 2023

Assim, se um atacante alterar o conteúdo de qualquer um dos ficheiros existentes no APK, seja o código, nome, versão das bibliotecas, etc., ao tentar instalar a aplicação, a mesma irá falhar.

AVALIAR MECANISMO DE ASSINATURAS DE FICHEIROS DO APK O objetivo do teste efetuado é avaliar a fiabilidade do mecanismo de assinaturas que previne a modificação *ad-hoc* de ficheiros num APK. Existem diversas formas de realizar o teste anteriormente descrito. Uma das operações realizadas no âmbito do teste foi a alteração ao conteúdo do ficheiro `/assets/button-check.json` existente no APK, seguida da tentativa de instalação da aplicação modificada. Para a realização deste tipo de teste, não é necessário a compilação e assinatura da aplicação, tendo sido feita uma alteração direta de um dos valores, sem efetuar qualquer tipo de assinatura à posteriori do APK da aplicação.

```
┌──(luis㉿luis)-[~/…/Dissertacao/Testes/ReverseEngineering/mecanismoAssinatura]
└─$ adb -s 424ac404 install Bolt_52.1_Removido_Xiaomi.apk
Performing Streamed Install
adb: failed to install Bolt_52.1_Removido_Xiaomi.apk: Failure [INSTALL_PARSE_FAI
LED_NO_CERTIFICATES: Failed to collect certificates from /data/app/vmdl938215229
.tmp/base.apk: META-INF/BNDLTOOL.SF indicates /data/app/vmdl938215229.tmp/base.a
pk is signed using APK Signature Scheme v2, but no such signature was found. Sig
nature stripped?]
```

Figura 17: Teste: Instalar aplicação após modificação de recursos do APK

O comportamento encontrado é apresentado na figura 17. Pode-se ver que a instalação falhou, validando assim a fiabilidade do mesmo para o teste efetuado. Para melhor avaliar este mecanismo de segurança, foi ainda i) substituído um dos ficheiros por outro com o mesmo nome, ii) foi também feita a alteração do seu conteúdo e iii) novos ficheiros, tudo isto no APK da *Bolt*. Para cada um dos testes foram criados novos ficheiros, ocorreu sempre a deteção de qualquer uma das modificações referidas, estando assegurada a validade do mecanismo de assinaturas para cada um dos casos.

Conclui-se portanto que o mecanismo de segurança está construido de forma a garantir que qualquer alteração por um possível atacante aos ficheiros originalmente assinados pelo desenvolvedor, seja bloqueada durante a instalação da aplicação.

5.4.1 *Assinatura do APK*

A assinatura do APK é um mecanismo essencial para a segurança de dispositivos *Android*, garantindo a autenticidade da aplicação, e que a mesma foi assinada pelos

desenvolvedores oficiais da aplicação. A assinatura usada tem como base a chave privada, com um procedimento idêntico ao original de forma a garantir a validade da mesma, o mesmo é válido para novas versões e ou atualizações da mesma aplicação.

Na realização de testes que impliquem a alteração do código fonte original, e não haja acesso à chave privada original é necessário criar um novo certificado '*self-signed*' para que seja possível instalar a aplicação no DM. A Listagem 14 ilustra a criação de certificado digital '*self-signed*', o qual gera um par de chave público e privado, que será guardada na keystore. Este certificado foi criado através do utilitário Keytool.

Listagem 14: Criar um certificado local - self signed

```
$ keytool -genkey -v -keystore <nomeCert>.keystore -alias <nome> -keyalg RSA
↪  -keysize 2048 -validity 10000
```

Na posse do certificado pode-se então assinar o *APK* com o certificado criado anteriormente, conforme exemplificado na Listagem 15.

Listagem 15: Assinar APK com certificado local

```
$ jarsigner -verbose -sigalg SHA256withRSA (<algoritmoEncriptacao>) -digestalg
↪  <algoritmoDe>SHA1 -keystore <nomeCert>.keystore  <nomeApkAssinar>.apk
↪  <nome-alias>
```

Deve ser prestada principal atenção ao algoritmo de assinatura escolhido (através da *flag* `-sigalg`), conforme ilustrado na Listagem 15. Isto porque o algoritmo de assinatura deve corresponder à chave-privada do algoritmo que foi selecionado aquando da criação do certificado (*flag* `-keyalg`), conforme ilustrado na Listagem 14.

Podemos observar na Tabela 11 um resumo da correspondência das assinaturas possíveis do APK, para cada um dos algoritmos do certificado existente. Para ambientes de produção devem ser escolhidos algoritmos robustos, a sua escolha seja também reflexo de outros fatores, tais como o tamanho da chave utilizada e a existência de ataques conhecidos.

Tabela 11: Keytool - Algoritmos e assinaturas correspondentes

Fonte: https://download.java.net/java/early_access/loom/docs/specs/man/keytool.html

Algoritmo	**Tamanho da chave**	**Algoritmo de assinatura por omissão**
DSA	Qualquer tamanho	**SHA256withDSA**
RSA	< 624	**SHA256withRSA**
	≤ 7680	**SHA384withRSA**
	> 7680	**SHA512withRSA**
EC	< 512	**SHA384withECDSA**
	≥ 512	**SHA512withECDSA**
RSASSA-PSS	< 624	**RSASSA-PSS**(com SHA-256)
	≤ 7680	**RSASSA-PSS** (com SHA-384)
	> 7680	**RSASSA-PSS** (com SHA-512)
EdDSA	255	**Ed25519**
	448	**Ed448**
Ed25519	255	**Ed25519**
Ed448	448	**Ed448**

Informação do certificado digital Bolt Nesta secção analisa-se o certificado utilizado pela aplicação *Bolt*, explicitando algumas das informações mais relevantes que foram encontradas.

```
└─$ keytool -printcert -file BNDLTOOL.RSA
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatex
t=true
Owner: O=Mtakso OÜ
Issuer: O=Mtakso OÜ
Serial number: 51c97f6a
Valid from: Tue Jun 25 12:30:50 WEST 2013 until: Sat Jun 19 12:30:50 WE
ST 2038
Certificate fingerprints:
         SHA1: CF:5A:18:39:37:79:2E:02:B5:87:34:A1:35:73:31:31:C8:56:1C
:AB
         SHA256: A7:5C:63:72:A0:B6:7D:B0:16:86:B4:7D:F6:8C:91:51:6E:E1:
62:29:EE:C4:C0:C6:7D:35:5E:32:20:7C:66:17
Signature algorithm name: SHA1withRSA (weak)
Subject Public Key Algorithm: 1024-bit RSA key (weak)
Version: 3
```

Figura 18: Keytool: Leitura do certificado Bolt

A figura 18 mostra o conteúdo do certificado público da aplicação *Bolt*. A informação do certificado é considerada como pública nomeadamente a chave pública, gerada por um algoritmo assimétrico - *RSA*. A chave pública tem por objetivo a certificação da assinatura do APK, não servindo para efetuar assinaturas.

Listagem 16: Informação do certificado digital Bolt

Owner: MTAKSO OÜ/Taxify OÜ;
Certificado de chave pública: RSA;
Tamanho do módulo: 1024 bits;
Validade: Até 19 de Junho de 2038;
Algoritmo de Hash: SHA1;
Assinatura: SHA1withRSA.

Podemos verificar na Listagem 16 que o dono (*Owner*) do certificado pertence à companhia *MTAKSO OÜ/Taxify OÜ*. Tal se deve ao facto de ser esse o nome original da empresa, tendo sido posteriormente alterado para *Bolt*.

O algoritmo de *hash* utilizado é o *SHA1withRSA*, sendo relevante perceber de que forma este é aplicado. Em primeiro lugar é utilizada a informação do certificado com a função de *Sha1* para transformar a informação do certificado numa sequência de letras e números designado por *digest*. Em seguida é aplicada ao resultado (*digest*) a chave privada, que neste caso faz parte do algoritmo *RSA*. O resultado

deste processo oferece a garantia de integridade e autenticidade da assinatura ao certificado, podendo o mesmo ser verificado através da chave pública da aplicação.

A função criptográfica, *SHA1withRSA* apresenta limitações em termos de segurança. Existem ataques que conseguiram provocar a colisão (*SHA1*), nos quais dois *inputs* distintos resultaram num mesmo resultado *(output)* (**Marc2017**).

É aconselhado o uso de algoritmos de *hash* mais seguros, tais como *SHA2* em vez de *SHA1*. O certificado de chave pública *RSA* de 1024 bits também é considerado como fraco, devendo ser empregue uma chave com pelo menos 2048 bits.

5.5 ANÁLISE DE CÓDIGO

Ao observar o código fonte extraído pelas ferramentas de descompilação, podemos constatar que o mesmo se encontra ofuscado. A análise seguidamente descrita ao código fonte foi feita seguindo o modelo de análise *black-box* sobre o código ofuscado.

Mesmo sem acesso ao código fonte original, existem nuances no código que permitem descobrir qual a linguagem em uso. Um desses casos é a presença da palavra *"Intrinsics"*, ou através da importação das bibliotecas associadas a *Kotlin*, sendo o uso desta linguagem identificável em diferentes parte do código.

A aplicação *Bolt* utiliza a *framework* Router, Interactor and Builder (RIB)s, para estabelecer uma ligação *VoIP*, com o condutor. A *framework* de arquitetura RIB [5], foi criada pela equipa de software da *Uber*, sendo utilizada por diversas aplicações móveis da *Uber*, sendo disponibilizado sob licença de código aberto. A *framework* RIB foi criada para ser utilizada em aplicações *Android* ou *iOS*, é modular e escalável.

5.5.1 *Verificação de rooted dispositivo*

A aplicação *Bolt* é iniciada através da classe *TaxifyApplication*. Esta classe efetua algumas verificações básicas antes de propriamente iniciar a aplicação, que vamos apresentar em seguida. Na prática, a aplicação valida as seguintes condições:

- Verifica se a versão do *SDK* é superior à versão 29;
- Verifica se o *Google Play Services* está disponível no dispositivo;

5 RIBs Github - Disponível em: https://github.com/uber/RIBs - Acedido a 24 de Nov. 2022

- Verifica a versão da aplicação (*Bolt*) instalada no dispositivo;

Nas verificações iniciais não existe nenhuma que tente identificar se o dispositivo se encontra *rooted* ou não, antes de iniciar a aplicação. Como tal é possível iniciar a aplicação, num DM *rooted* sem ser encontrada nenhuma limitação de uso.

Existe no código-fonte um método que efetua a verificação por existência de *root*. Este método encontra-se na classe `(com.veriff.sdk.internal.h7)`, sendo este apenas utilizado durante tarefas de *log*.

5.5.2 *Mensagens de Log e Bug reports*

A aplicação gera diversa informação que é escrita nos *logs* do sistema. Ao observar o código fonte foi possível identificar que é escrito nos *logs* informação relativa aos tempos de inicialização da aplicação, através da classe `eu.bolt.client.helper.e`.

Além dos *logs* gerados e que podem ser obtidos através do *logcat*, juntamente com as mensagens de *log* do sistema, observa-se que a aplicação utiliza o *FirebaseCrashlytics* para recolha de relatórios de erros. O *FirebaseCrashlytics* é um serviço de agregação de relatórios de falhas e funcionamento gerados pela aplicação, que permite centralizar toda essa informação em servidores da *Firebase* na *Cloud*, geridos pela *Google* numa única plataforma.

As figuras 19 e 20 representam, respetivamente as definições locais do *Crash Analytics* e um exemplo de ficheiro *report* do *Crash Analytics*. Os conteúdos representados nessas figuras foram obtidos com a ferramenta *Objection* (secção 4.3.3), tirando partido do facto do *Objection* criar uma cópia temporária dos ficheiros locais do dispositivo.

Ao consultar o ficheiro `com.crashlytics.settings.json` que contém as definições do ficheiro *Firebase crashlytics*, conforme ilustrado pela figura 19, observa-se que a configuração define a recolha de erros e não guarda informação de comportamento (*analytics*). Também podemos observar na mesma figura 19 que a informação recolhida não está a ser enviada para a *Firebase*.

É possível confirmar através do campo `jailbroken:false` do relatório (path: `/data/user/0/ee.mtakso.client/files/.com.google.firebase.crashlytics.files.v2:ee.mtakso.client/open-sessions/6570A3BB006100011BABC685A1727FCB/report`), apresentado na figura 20 que a aplicação não identificou o dispositivo como estando *rooted*. Esta falha na correta identificação representa uma vulnerabilidade, um dispositivo *rooted* pode por em causa o correto funcionamento

```
ee.mtakso.client on (Xiaomi: 9) [usb] # cd .com.google.firebase.crashlytics.files.v2:ee.mtakso.client
/data/user/0/ee.mtakso.client/files/.com.google.firebase.crashlytics.files.v2:ee.mtakso.client
ee.mtakso.client on (Xiaomi: 9) [usb] # ls
Type       Last Modified              Read  Write  Hidden  Size     Name
─────────  ─────────────────────────  ────  ─────  ──────  ───────  ──────────────────────────────
Directory  2023-12-06 14:57:40 GMT  True  True   False   4.0 KiB  open-sessions
Directory  2023-11-16 17:17:08 GMT  True  True   False   4.0 KiB  reports
Directory  2023-04-25 21:53:57 GMT  True  True   False   4.0 KiB  priority-reports
Directory  2023-10-21 01:49:42 GMT  True  True   False   4.0 KiB  native-reports
File       2023-12-05 16:33:06 GMT  True  True   False   711.0 B  com.crashlytics.settings.json

Readable: True  Writable: True
ee.mtakso.client on (Xiaomi: 9) [usb] # file cat com.crashlytics.settings.json
Downloading /data/user/0/ee.mtakso.client/files/.com.google.firebase.crashlytics.files.v2:ee.mtakso.client/com.crash
lytics.settings.json to /tmp/tmpz2lpz3_u.file
Streaming file from device...
Writing bytes to destination...
Successfully downloaded /data/user/0/ee.mtakso.client/files/.com.google.firebase.crashlytics.files.v2:ee.mtakso.clie
nt/com.crashlytics.settings.json to /tmp/tmpz2lpz3_u.file
═══
{"settings_version":3,"cache_duration":86400,"features":{"collect_logged_exceptions":true,"collect_reports":true,"co
llect_analytics":false,"prompt_enabled":false,"push_enabled":false,"firebase_crashlytics_enabled":false,"collect_anr
s":true,"collect_metric_kit":false,"collect_build_ids":true},"app":{"status":"activated","update_required":false,"re
port_upload_variant":2,"native_report_upload_variant":2},"fabric":{"org_id":"528b6c3b2a0ddbb6880000ae","bundle_id":"
ee.mtakso.client"},"on_demand_upload_rate_per_minute":10,"on_demand_backoff_base":1.2,"on_demand_backoff_step_durati
on_seconds":60,"app_quality":{"sessions_enabled":true,"sampling_rate":1,"session_timeout_seconds":1800},"expires_at"
:1701880386897}═══
```

Figura 19: Objection - Definições *CrashAnalytics*

```
ee.mtakso.client on (Xiaomi: 9) [usb] # file cat report
Downloading /data/user/0/ee.mtakso.client/files/.com.google.firebase.crashlytics.files.v2
:ee.mtakso.client/open-sessions/6570A3BB006100011BABC685A1727FCB/report to /tmp/tmpg8vryl
_x.file
Streaming file from device...
Writing bytes to destination...
Successfully downloaded /data/user/0/ee.mtakso.client/files/.com.google.firebase.crashlyt
ics.files.v2:ee.mtakso.client/open-sessions/6570A3BB006100011BABC685A1727FCB/report to /t
mp/tmpg8vryl_x.file
═══
{"sdkVersion":"18.2.11","gmpAppId":"1:718916732167:android:acb81c1cb838469f","platform":4
,"installationUuid":"579254f392a64fd9890ff20af1f4aca8","buildVersion":"2086","displayVers
ion":"CA.52.1","session":{"generator":"Crashlytics Android SDK/18.2.11","identifier":"NjU
3MEEzQkIwMDYxMDAwMTFCQUJDNjg1QTE3MjdGQ0I=","startedAt":1701880763,"crashed":false,"app":{
"identifier":"ee.mtakso.client","version":"2086","displayVersion":"CA.52.1","installation
Uuid":"579254f392a64fd9890ff20af1f4aca8"},"os":{"platform":3,"version":"9","buildVersion"
:"REL","jailbroken":false},"device":{"arch":9,"model":"Redmi Note 3","cores":6,"ram":2976
002048,"diskSpace":26283376640,"simulator":false,"state":0,"manufacturer":"Xiaomi","model
Class":"aosp_kenzo"},"generatorType":3}}═══
```

Figura 20: Objection - Resumo do ficheiro `report` para o *CrashAnalytics*

da aplicação, através de modificações indevidas tanto sobre a aplicação como dispositivo.

A restante informação apresentada na figura 20 obtida não é considerada como sensível, dado que é relativa ao *hardware* e *software* do dispositivo, utilizado para correr a aplicação, assim como as definições do *Google Firebase*, ilustradas na figura 19.

5.5.3 *Bibliotecas Nativas*

As bibliotecas nativas são por norma escritas em código C/C++ e compiladas para as diversas arquiteturas que podem ser usadas para correr a aplicação. A sua função é optimizar tarefas de baixo nível, que tenham um papel crítico, como processamento de imagens, áudio e criptografia. As bibliotecas nativas são principalmente relevantes no contexto de segurança, em aplicações que utilizam as mesmas para tarefas de criptografia. Muitas vezes o código nestas não é analisado e como tal podem conter *hardcoded secrets*[6].

Lista das bibliotecas nativas em uso pela aplicação *Bolt*:

- Bibliotecas de Imagens:
  - `libbarhopper_v3.so`;
  - `libimage_processing_util_jni.so`
- Biliotecas de relatórios e logs:
  - `libcrashlytics.so`;
  - `libcrashlytics-trampoline.so`;
  - `libcrashlytics-common.so`;
  - `libcrashlytics-handler.so`;
- Biblioteca para chamadas em tempo real:
  - `libsinch-android-rtc.so`

No caso da bilioteca `libbarhopper_v3.so` não existia documentação sobre o seu funcionamento, como tal foi necessário recorrer à ferramenta de engenharia reversa conhecida como *Ghidra*[7], para poder obter mais informação sobre o seu funcionamento.

Podemos ver através da figura 21, a implementação de uma função em C/C++ que parece estar relacionada com diversas tarefas para processamento de imagens *bitmap* em *Android*.

A ferramenta ADB, revela a existência de bibliotecas definidas como opcionais, conforme observado pela Listagem 17. Isto significa que mesmo se as bibliotecas

6 Hardcoded Cryptographic Secrets - Google Developers- Disponível em: https://developer.android.com/privacy-and-security/risks/hardcoded-cryptographic-secrets - Acedido a 20 Fev. 2024

7 Github Ghidra, Disponível em: https://github.com/NationalSecurityAgency/ghidra - Acedido a 10 Mar. 2023

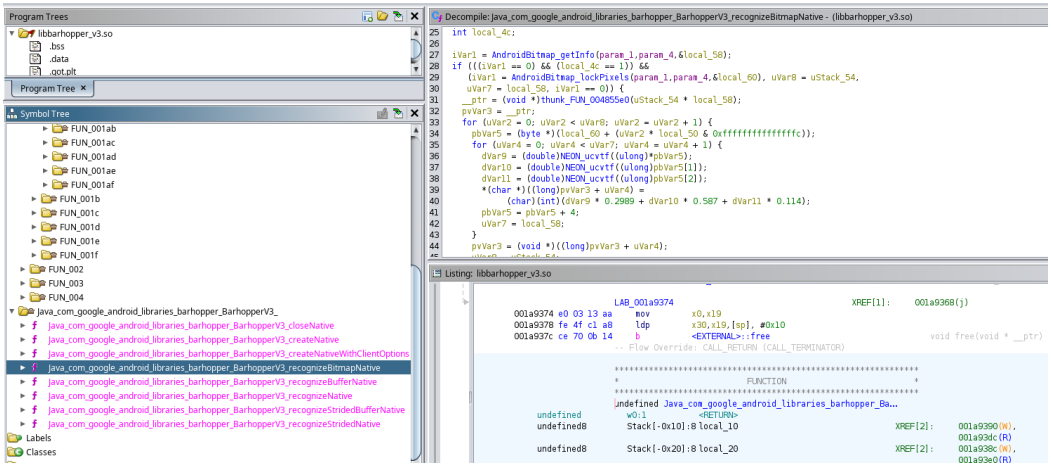


Figura 21: Ghidra - Funções da biblioteca `libbarhopper_v3`

não estiverem disponíveis a aplicação continua o seu funcionamento, no entanto as suas funcionalidades serão reduzidas.

Bibliotecas opcionais *Bolt*:

Listagem 17: Bibliotecas para funcionalidades consideradas como Opcionais

```
$ adb shell dumpsys package ee.mtakso.client
    ...
    (Omitido para brevidade)
    ...
  usesOptionalLibraries:
      org.apache.http.legacy
      androidx.window.extensions
      androidx.window.sidecar
  usesLibraryFiles:
      /system/framework/org.apache.http.legacy.boot.jar
```

As bibliotecas `androidx.window.extensions` e `androidx.window.sidecar` são responsáveis pela gestão de aplicações que suportem dispositivos com ambientes de múltiplas janelas[8]. A biblioteca `org.apache.http.legacy` é responsável por oferecer compatibilidade para sistemas Android 6.0 e anteriores que necessitam de utilizar o client *Apache Http*[9].

Em qualquer um dos casos que foram descritos anteriormente, nenhuma das bibliotecas nativas está associada a operações de criptografia sobre dados sensíveis ou privados da aplicação. Dito isto a sua relevância para o contexto deste trabalho não se mostrou significativa para que fosse aprofundado o seu funcionamento e/ou conteúdo.

8 Extensões do WindowManager - Disponível em: https://source.android.com/docs/core/display/windowmanager-extensions?hl=pt - Acedido a 05 Nov. 2023

9 Apache deprecation - Disponível em: https://developer.android.com/about/versions/pie/android-9.0-changes-28 - Acedido a 4 Nov. 2023

5.6 ANÁLISE AO ANDROID MANIFEST

A análise ao *Android Manifest* é um dos pontos iniciais da análise à segurança de uma aplicação móvel, de forma a compreender o funcionamento da aplicação. A sua análise torna-se de especial importância no caso em prática, pois não existe acesso ao código fonte original e o mesmo está ofuscado, o que torna o processo de *reverse engenieer* mais complexo e demorado.

O package Bolt contém os seguintes elementos:

- **minSDK Version:** 21 (*Android* 5.0); *TargetSDK Version*: 32 (*Android* 13.0);
- **Linguagens utilizadas:** *Kotlin, Java*;
- **Activity inicial:** *ee.mtakso.client.activity.SplashHomeActivity*;

Devido à quantidade de informação disponível no ficheiro `AndroidManifest.xml`, a sua análise encontra-se dividida em cinco subsecções, para que seja mais fácil ao leitor seguir os passos tomados.

5.6.1 *Permissões*

Uma descrição exaustiva sobre as características e importância das permissões para a segurança do DM em *Android* pode ser consultada na Secção 3.8.2, página 45.

Esta secção serve para analisar as permissões que foram encontradas no ficheiro `AndroidManifest.xml` da aplicação Bolt Versão 52.1, e perceber o possível risco que estas podem representar para o seu utilizador.

O risco indicado na Tabela 12, corresponde ao nível por omissão definido pela *Google*[10], a menos que explicitamente alterado pelos criadores da aplicação.

Para as permissões com nível de risco normal ou de assinatura, não é necessário que o utilizador autorize o uso das mesmas. As permissões perigosas ou especiais/privilegiadas, requerem autorização explícita do utilizador. Assim, é importante perceber se o seu nível foi alterado pelo criador da aplicação como a introdução de permissões indevidas/excessivas ou alteradas como forma de introduzir um risco acrescido ao seu utilizador.

10 Descrição das permissões - *Android* Developers: https://developer.android.com/reference/android/Manifest.permission - Acedido a 10 de Nov. 2022

Tabela 12: Permissões solicitadas, AndroidManisfest.xml (Bolt 52.1)

Permissão	Descrição	Risco
ACCESS_FINE_LOCATION	Localização GPS com alto grau de precisão	Perigoso
ACCESS_COARSE_LOCATION	Localização de rede móvel com menor grau de precisão	Perigoso
CAMERA	Permite capturar fotos e vídeos	Perigoso
RECORD_AUDIO	Permite gravar áudio	Perigoso
POST_NOTIFICATIONS	Permite envio de notificações através dos canais de notificação disponíveis.	Perigoso
ACCESS_NETWORK_STATE	Permite observar o estado da rede (em ligação, ligado, desligado) (Acesso ao ConnectivityManager)	Normal
ACCESS_WIFI_STATE	Gerir todas as funções de acesso WiFi (Acesso WifiManager)	Normal
BLUETOOTH	Criar Ligações Bluetooth com dispositivo emparelhados	Normal
FOREGROUND_SERVICE	Permite à aplicação executar uma tarefa em *foreground* e notificar o mesmo através do (*status bar*)	Normal
GET_PACKAGE_SIZE	Obtém o espaço ocupado pela aplicação	Normal
INTERNET	Permite à aplicação realizar operações de troca de dados através da rede	Normal
MODIFY_AUDIO_SETTINGS	Permite à aplicação alterar as definições de áudio	Normal
NFC	Permite comunicar através de NFC (*tags*, cartões, leitores)	Normal
RECEIVE_BOOT_COMPLETED	Permite iniciar de forma automática após o *boot* do sistema	Normal
USE_FULL_SCREEN_INTENT	Utilizado para notificações que requerem uma atenção imediata do seu utilizador	Normal
VIBRATE	Permite controlar a vibração do dispositivo	Normal
WAKE_LOCK	Previne o dispositivo de entrar em suspensão	Normal
RECEIVE	Permite troca de mensagens na Cloud (Google Cloud Messaging - GCM, Firebase cloud Messaging - FCM)	Assinatura
BIND_GET_INSTALL_REFERRER_SERVICE	Identifica a origem da instalação, requerida pela *Firebase*	Normal
AI_ID	Permissão atribuída automaticamente[11], para *Android* 13 e superior, utilizada para monetizar anúncios personalizados	Não atribuído

Além das permissões referidas anteriormente existe também a permissão específica que se encontra descrita na Listagem 18.

Listagem 18: XML: Permissão personalizada específica (Bolt 52.1)

```
1    <permission android:label="Get Veriff status reports"
     ↪  android:name="ee.mtakso.client.VERIFF_STATUS_BROADCAST_PERMISSION"
     ↪  android:protectionLevel="signature"/>
```

Permissões excessivas

Das permissões descritas na Tabela 12 como perigosas, podemos referir que a permissão de acesso à câmara (`android.permission.CAMERA`) e ao microfone (`android. permission.RECORD_AUDIO`) podem ser consideradas como excessivas. A permissão de acesso ao microfone é necessária se pretendermos realizar uma

chamada Voice over Internet Protocol (VoIP) através da aplicação com o condutor[12]. No caso da permissão de acesso à câmera do DM não existe nenhuma funcionalidade anunciada que necessite do uso da mesma na aplicação *Bolt*. A consulta ao documento da Política de Privacidade, abordado na Secção 5.1, também não revela qualquer referência para o tratamento de dados pessoais ligados a vídeos ou imagens recolhidas.

Foi verificado que a permissão de acesso ao microfone não é requerida durante a instalação, nem durante o uso da aplicação enquanto não foi necessário entrar em contacto com o condutor. O mesmo pode ser observado ao ler de forma dinâmica sobre as permissões que foram concedidas à aplicação, através do comando presente na Listagem 28 (página 132).

LOCATION ACCESS A permissão de acesso à localização é necessária para o uso básico da aplicação, para encontrar trotinetas partilhadas nas imediações, além de facilitar o processo de agendamento do transporte, entre cliente e condutor.

A restrição de acesso à localização através de Wi-Fi, bem como à limitação de acesso de informação sobre a ligação em uso (informação de `SSID` e `BSSID`), foram alterações impostas a partir de *Android* 9 (API lvl 28). Esta alteração torna necessário requerer as permissões que foram encontradas no `Manifest`, que são `ACCESS_FINE_LOCATION` ou `ACCESS_COARSE_LOCATION` e `ACCESS_WIFI_STATE`.

A localização é também um importante fator de segurança do próprio cliente e do condutor de forma a definir o trajeto que será percorrido, ou como fator de prova criminal sob a forma de geolocalização temporal, em diferentes processos criminais/judiciais (acidente, assalto, cobrança indevida, violência física e/ou sexual).

Consideramos que as permissões requeridas para a localização foram corretamente estabelecidas, sendo as mesmas necessárias do nosso ponto de vista. No entanto tem que ser tido em conta que da forma como está configurada, a aplicação *Bolt* obtém a localização com a maior precisão possível (`ACCESS_FINE_LOCATION`), conseguindo desta forma mapear com precisão toda a localização do seu utilizador.

5.6.2 *Content Provider URI*

O sistema de permissões tradicional não é suficiente quando estamos a lidar com *content providers*.

12 Chamadas VoIP, através da App Bolt - Disponível em: https://bolt.eu/en-ke/blog/feature-update-make-calls-through-the-bolt-app-at-no-extra-cost/ - Acedido a 07 Dez. 2023

Um *content provider* pode querer limitar as permissões de leitura e/ou de escrita, de forma a se proteger ao usar URI's para obter informações. No caso de estudo em apreço foi encontrado o atributo `android:grantUriPermissions="true"` no `manifest` o qual indica, que a aplicação concede acesso temporário a arquivos específicos da aplicação, como podemos observar pela Listagem 19. Tal permite outras aplicações possam aceder a documentos criados pela aplicação, tais como imagens ou documentos.

O atributo `android:name="androidx.core.content.FileProvider"` é utilizado para declarar o componente *FileProvider* que será usado para gerir o acesso temporário a um arquivo solicitado. *FileProvider* é um tipo 'especial' de *ContentProvider* que permite a uma aplicação partilhar dados com outras aplicações de maneira segura e protegida, sem dar no entanto acesso direto ao sistema de ficheiros.

Juntos, esses atributos permitem que outras aplicações possam aceder aos arquivos na pasta de dados da aplicação sem no entanto dar acesso a toda a pasta. Isso ajuda a proteger os dados da aplicação e evita que outras aplicações possam aceder ou modificar arquivos aos quais não deveriam ter acesso.

Listagem 19: Permissão: *Content Provider* aos ficheiros definidos no ficheiro *file_paths*

```
1    <provider android:authorities="ee.mtakso.client.images.fileprovider"
     ↪  android:exported="false" android:grantUriPermissions="true"
     ↪  android:name="androidx.core.content.FileProvider">
2    <meta-data android:name="android.support.FILE_PROVIDER_PATHS"
     ↪  android:resource="@xml/file_paths"/>
3    </provider>
```

Ao observar a Listagem 20 que corresponde a um excerto do ficheiro `file_paths.xml` é possível constatar que os ficheiros aos quais se dá acesso têm no seu conteúdo informação resultante do *debug* da aplicação.

Listagem 20: Conteúdo do ficheiro: file_paths.xml

```
1    <paths>
2    <external-cache-path name="cache" path="."/>
3    <files-path name="bug_report" path="servicedesk"/>
4    </paths>
```

O ficheiro `bug_report` não existia na *path* assinalada na Listagem 20, no entanto esse não era o objetivo desta secção. O importante aqui a reter é a importância do uso de um outro modelo de permissões aplicadas aos URI, que reduza as permissões necessárias, limitando/restringindo o seu comportamento.

5.6.3 *Exported Content Providers*

No seguimento da secção de Content Providers, falta ainda destacar a existência de um único *Exported Content Provider* na aplicação *Bolt* `com.facebook.FacebookContentProvider`. Este é utilizado para troca de informação com a API do *Facebook*.

A sua função aqui encontrada é destinada a garantir a autenticação do utilizador, ao associar uma conta do *Facebook* previamente criada à conta em uso da *Bolt*. Esta função é descrita como estabelecer o *link*, entre ambas as contas.

5.6.4 *Exported Activities*

Num total de 25 *activities* existentes, seis estão marcadas como `exported="true"` no `manifest`, estas são apresentadas na Tabela 13. Note-se que por omissão as *activities* estão marcadas com a *flag* a *false*.

Existem duas formas de se declarar uma *Activity* como *exported*, seja com o uso da *flag* `exported="true"`, ou através do uso da instrução `<intent-filter>`, ambas descritas no ficheiro `AndroidManifest.xml`. Ambas permitem interagir com o componente (*activity*), partindo de uma aplicação terceira. Permitindo assim a aplicações externas interagir com o componente, se não existirem as devidas permissões que impeçam a sua interação. Tal obriga a um cuidado especial, referente às interações que são executadas entre os referidos componentes.

Apesar de parecer contraproducente utilizar uma *exported Activity*, existem razões que levaram à sua criação. Podemos destacar a integração/usabilidade com outras aplicações, nomeadamente na exportação de uma funcionalidade ou recurso da nossa aplicação que seja considerada como público(a), no qual se destacam componentes criados explicitamente para serem reutilizáveis ou públicos por outras aplicações.

Existem no entanto cuidados que devem ser tidos ao analisar uma *exported Activity*. Desde logo perceber se as suas funcionalidades se adequam ao objetivo da criação, e ajustar de acordo com o definido.

Um dos exemplos que podemos referir de como é possível lançar a aplicação *Bolt* partindo de uma aplicação terceira pode ser encontrado na listagem 21.

Tabela 13: **Exported Activities**, AndroidManisfest.xml (Bolt 52.1)

Classe	Descrição
`ee.mtakso.client.activity.SplashHomeActivity`	Responsável por abrir o mapa que representa a página principal da aplicação.
`ee.mtakso.client.newbase.RideHailingMapActivity`	Abre o mapa, apresentado as moradas e destinos mais utilizados.
`ee.mtakso.client.newbase.link.DeeplinkActivity`	Gere a inicialização da aplicação, através de *deeplinks*.
`ee.mtakso.client.newbase.deeplink.AppsFlyerDeeplinkActivity`	Permite obter conteúdo personalizado com base no contexto do seu utilizador.
`com.adyen.threeds2.internal.p104ui.activity.ChallengeActivity`	Package responsável pelo processamento de pagamentos
`com.facebook.CustomTabActivity`	Inicializa uma *WebView* personalizada da aplicação *Facebook*

Listagem 21: Iniciar a aplicação a partir de uma app terceira (*Exported Activity*)

```
1 Intent intent = new Intent();
2 intent.setComponent(new ComponentName("ee.mtakso.client",
  ↪  "ee.mtakso.client.newbase.RideHailingMapActivity"));
3 startActivity(intent);
```

Foram executados testes a todas as *exported activities* anteriores de forma a garantir que as mesmas não permitissem aceder ou revelar informação sensível/pessoal do seu utilizador. Para esse efeito foi utilizada a ferramenta Drozer, com o intuito de simplificar o processo de inicialização das diversas *Activities* existentes, conforme apresentada na figura 22.

```
└─$ docker run --net host -it withsecurelabs/drozer console connect --server 192.168.1.67
Selecting 3afaaf8536b50d4f (Xiaomi Redmi Note 3 9)

            ..                    ..:.
           ..o..                  .r..
            ..a..  . ..... .  ..nd
              ro..idsnemesisand..pr
              .otectorandroidsneme.
           .,sisandprotectorandroids+.
         ..nemesisandprotectorandroidsn:.
        .emesisandprotectorandroidsnemes..
      ..isandp,..,rotecyayandro,..,idsnem.
      .isisandp..rotectorandroid..snemisis.
      ,andprotectorandroidsnemisisandprotec.
     .torandroidsnemesisandprotectorandroid.
     .snemisisandprotectorandroidsnemesisan:
     .dprotectorandroidsnemesisandprotector.

drozer Console (v2.5.0)
dz> run app.activity.start --component ee.mtakso.client ee.mtakso.client.newbase.RideHailingMapActivity
```

Figura 22: Inicializar *Exported Activity* - *Drozer*

Na aplicação *Bolt* podemos encontrar uma situação que é bastante comum: As *Activities* que iniciam, ou que são a 'porta' de entrada para uma aplicação, podem

ser inicializadas por outras aplicações. Este é um comportamento que não interfere com a segurança da aplicação, isto porque os dados apresentados pela *Activity* iniciada requerem validação, seja no caso de pagamento para marcação de uma viagem, ou para iniciar a sessão pelo *Facebook*. No entanto existe a informação dos destinos mais utilizados (viagens efetuadas).

Exported activities podem também ser utilizadas para redirecionar o utilizador para um *link*, através de *deeplinks* indevido. Mais informação sobre o que são *deeplinks* e como podem ser usados por atacantes encontram-se disponíveis em 7.3, página 155.

Outro caso que encontramos nesta aplicação específica é que não existe autenticação do seu utilizador para aceder à aplicação. O que nos permite ter acesso a todas as funções da aplicação e do seu utilizador, se o mesmo tiver criado a conta previamente. A aplicação suporta uma conta por utilizador, com a excepção no caso de clientes empresariais, que podem associar múltiplos funcionários[13]. No entanto nada impede o cliente de utilizar múltiplas contas no mesmo dispositivo móvel. O registo pode ser realizado de duas formas que são:

- Número de telefone;
- Associação de conta *Google*;

As restrições/validações de uso da aplicação são impostas aquando o pagamento da viagem. No entanto um possível atacante, apenas por iniciar a aplicação pode obter acesso a informações como locais favoritos, viagens efetuadas, conversas trocadas, número de telefone e endereço de email, com uma simples consulta ao perfil do utilizador. Este caso de uso é partindo do pressuposto que existe acesso ao DM sem bloqueio de ecrã do mesmo.

5.6.5 *Exported Services*

Existem considerações que devem ser tomadas quando pretendemos identificar vulnerabilidades que possam existir ao analisar serviços numa aplicação *Android*. Serviços podem incorretamente ser acedidos por qualquer aplicação que possa ter permissão para o fazer, o que pode originar acesso indevido de dados ou funcionalidades.

13 Gestão de contas, Bolt - Disponível em:https://bolt.eu/en/support/articles/45861/ - Acedido a 9 Jan. 2024

Foram encontrados na aplicação *Bolt* quatro serviços que são exportados, os quais encontram-se apresentados na Tabela 14. A sua identificação mais uma vez provém do uso da *flag: exported="true"*.

Tabela 14: **Exported Services**, AndroidManisfest.xml (Bolt 52.1)

Classe	Descrição
`com.google.android.play.core.assetpacks.AssetPackExtractionService`	Serviço da Google Play Core Library, para gestão e extração de pacotes de arquivos (*asset packs*).
`com.google.android.gms.auth.api.signin.RevocationBoundService`	Revogação de *tokens* de autenticação em aplicações que utilizam a API de *login Google*[14].
`com.sinch.android.rtc.internal.client.fcm.InstanceIDTokenService`	Apesar de declarada, esta classe não se encontra presente entre os ficheiros descompilados.
`androidx.work.impl.background.systemjob.SystemJobService`	Serviço do *WorkManager* para execução de tarefas em segundo plano do *Android*[15].

Quando um serviço é exportado sem qualquer restrição de permissão, qualquer aplicação maliciosa ou não, pode aceder e utilizar as funções por este implementadas.

Nas classes apresentadas na Tabela 14, a classe pertencente à *Sinch* não existe no código descompilado, como tal não é possível analisar o seu código. No entanto, para as restantes classes, nomeadamente a classe *RevocationBoundService*, serviço este que lida apenas com ações muito específicas como verificar a validade do *token* de autenticação da *Google*, se o utilizador revogar o acesso será iniciado um serviço para limpar dados relativos à sua autenticação. Os restantes serviços são utilizados para gestão, download e extração de conteúdos pela *Google Play Core*, com exceção do serviço para a criação de tarefas (*jobs*), assente na API do *WorkManager*.

5.6.6 *Exported Broadcast Receivers*

Existe um fator que torna os *broadcast receivers* distinto de todos os anteriores. Falamos da possibilidade do seu uso, sem existir declaração do mesmo no ficheiro `AndroidManifest.xml`. Fator este que é obrigatório em todos os restantes componentes, nos quais a não declaração impossibilita o uso dos mesmos tanto pela própria aplicação como pelo sistema *Android*.

Esta característica torna-os suscetíveis a introduzir riscos de segurança, os quais podem incluir componentes "escondidos", dado não serem explicitamente declarados, apenas podem ser acedidos pela própria aplicação, uma vez que não existe forma de outras aplicações identificarem a existência do mesmo.

Uma falha de segurança que pode ocorrer ao criar um *broadcast receiver* de forma dinâmica, é a inexistência de permissões que limitem o uso do mesmo. Tal pode originar acesso indevido a dados, bem como o desempenho de ações privilegiadas sem possuir privilégios para tal.

Na aplicação *Bolt* existem quatro *Broadcast Receiver*, descritos na Tabela 15.

Tabela 15: **Exported BroadcastReceiver**, AndroidManisfest.xml (Bolt 52.1)

Classe	**Descrição**
`com.appsflyer.MultipleInstallBroadcastReceiver`	Lida com informação acerca da origem da instalação e execução de ações personalizadas, geridas pela *AppsFlyer*.
`com.google.firebase.iid.FirebaseInstanceIdReceiver`	Recebe notificações de instâncias do Firebase Cloud Messaging (FCM) relacionadas com *Firebase Instance ID*.
`com.adyen.threeds2.internal.AppUpgradeBroadcastReceiver`	Lida com transmissões relacionadas a atualizações de aplicações no contexto do 3D Secure 2 da Adyen.
`androidx.work.impl.diagnostics.DiagnosticsReceiver`	Recebe diagnósticos relacionados ao *WorkManager*, uma biblioteca para gestão de tarefas agendadas em segundo plano.

Com este capítulo foi dado a conhecer ao leitor métodos de análise estática aos ficheiros locais que fazem parte do APK. Ficou registada a abordagem utilizada para uma análise mais profunda ao ficheiro AndroidManifest.xml e de como este pode ser útil numa análise estática. No próximo capítulo serão apresentados os artefactos forenses que foram identificados em todos os espaços em uso pela aplicação *Bolt* no dispositivo de testes.

6

ARTEFACTOS FORENSES NUMA ANÁLISE *POST MORTEM*

Este capítulo foca os principais artefactos forenses da aplicação Bolt que ocorrem aquando de uma análise *Post-Mortem*.

6.1 VARIÁVEIS DE AMBIENTE

É importante conhecer as variáveis de ambiente e as *paths* dos diretórios mais relevantes para uma análise dos artefactos gerados.

Recorrendo à ferramenta *Objection*, é possível obter o acesso a essa mesma informação, conforme podemos observar através da Tabela 16.

Tabela 16: Variáveis de Ambiente (Bolt 52.1)

Nome	**Path**
Cache Directory	`/data/user/0/ee.mtakso.client/cache`
Code Cache Directory	`/data/user/0/ee.mtakso.client/code_cache`
External Cache Directory	`/storage/emulated/0/Android/data/ee.mtakso.client/cache`
Files Directory	`/data/user/0/ee.mtakso.client/files`
obbDir	`/storage/emulated/0/Android/obb/ee.mtakso.client`
Package Code Path	`/data/app/ee.mtakso.client-a2XGAVyASjgXv-H_tx2sqg/`

6.2 DIRETÓRIOS RELEVANTES NUM DISPOSITIVO ANDROID

O SO *Android* disponibiliza a uma aplicação vários espaço próprios no sistema de ficheiros para o armazenamento de dados em ficheiros. Esses espaços organizam-se em espaço público e espaço privado. De seguida, analisar-se-ão ambos os espaços na perspetiva forense da aplicação *Bolt*.

6.2.1 *Espaço Público*

Como o nome sugere, o espaço público não requer privilégios de administrador, significando que os ficheiros existentes podem ser acedidos de forma direta, mesmo que o dispositivo não tenha sofrido o processo de *root*.

O caminho do espaço público da aplicação *Bolt* é `/sdcard/Android/data/ee.mtakso.client`, com a última parte – `ee.mtakso.client` – a refletir o nome do *package* da aplicação.

A inexistência da opção `uses-permissionandroid:name="android.permission.WRITE_EXTERNAL_STORAGE"` no `AndroidManifest.xml` indica que a aplicação não pode escrever ou ler para dispositivos de armazenamento externos.

No caso da aplicação *Bolt*, não foi possível obter dados com valor forense que estivessem na diretoria `/sdcard` e diretorias descendentes. Podemos ver através da figura 23, a composição do espaço público da aplicação *Bolt* e as suas diretorias descendentes, foi utilizada uma cópia integral da mesma para propósitos de demonstração.

```
┌──(luis㉿luis)-[~/…/sdcard/Android/data/ee.mtakso.client]
└─$ tree -d -L 2
.
├── cache
│   └── debug
└── files
```

Figura 23: Estrutura do espaço público

Com alguns dados, mas de difícil interpretação, tem-se a diretoria `"cache"`. Nesta pasta e subpasta, os ficheiros presentes estão associados à navegação, atuando como *cache* de mapas e respetivos elementos, como estradas, declives de terreno e pontos de interesse. Parte do conteúdo dos ficheiros representam dados binários sobre os quais não existe representação/conversão para representação textual ou para uma representação passível de fácil interpretação humana. Não obstante, foi possível consultar de forma legível o conteúdo do ficheiro `cache_r.0`, onde existem referências para o uso de URL's para acesso a ícones de marcação de localizações no mapa. Podemos observar um excerto na figura 24, no entanto estes não revelam qualquer localização geográfica do seu utilizador.

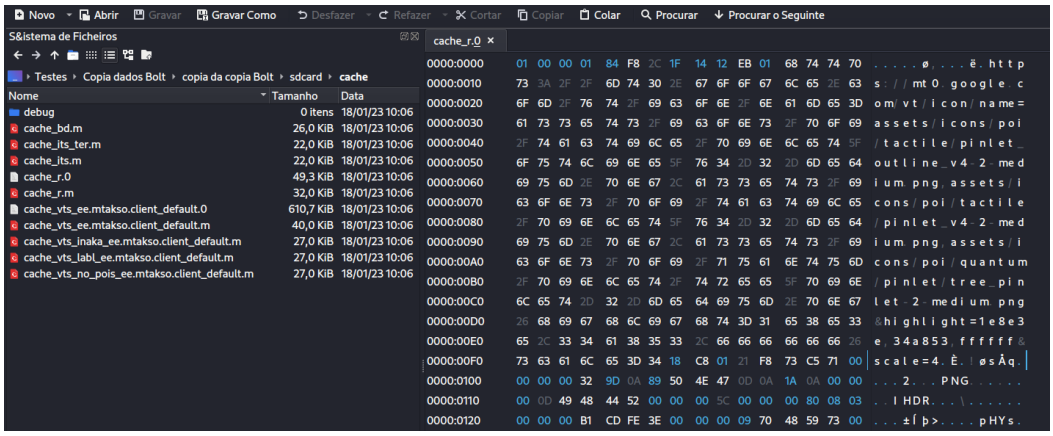


Figura 24: Excerto legível do ficheiro `cache_r.0`, onde podemos observar o *url* para a imagem.

6.2.2 *Espaço Privado*

O espaço privado de uma aplicação, localizado numa subpasta da diretoria `/data/data/` apenas pode ser acedido pela própria aplicação ou num dispositivo que se encontre em modo "*root*". A figura 25 mostra a hierarquia de diretórios da aplicação *Bolt* com origem na pasta `/data/data/ee.mtakso.client`.

```
kenzo:/data/data/ee.mtakso.client # find . -maxdepth 2 -type >
.
|cache
├──instrument
├──com.google.android.gms.maps.volley
├──WebView
|code_cache
|files
├──splitcompat
├──.com.google.firebase.crashlytics.files.v2:ee.mtakso.client
├──AFRequestCache
├──phenotype
├──phenotype_storage_info
├──sinch
|shared_prefs
|databases
|no_backup
|app_webview
├──Default
├──BrowserMetrics
|app_textures
```

Figura 25: Estrutura do espaço privado - `/data/data/ee.mtakso.client`

Na pasta `/data/` existem dois diretórios principais – "`user`" e "`user_de`". O diretório "`user`" guarda os dados, as preferências e definições do utilizador principal do dispositivo. De facto, o modelo de utilização do *Android* suporta a existência de

um utilizador principal e a possibilidade de existência de utilizadores secundários, i.e., outras contas de utilização.

Conforme descrito no Capítulo 4, o dispositivo de testes contém apenas um único utilizador, existe um link simbólico do conteúdo da diretoria `/data/user/` para a `/data/data/`, podemos observar essa situação a partir da figura 26.

```
kenzo:/data/user # ls -l
total 0
lrwxrwxrwx 1 root root 10 1976-08-28 14:42 0 → /data/data
```

Figura 26: Link simbólico do espaço privado

Por sua vez o diretório "`user_de`", é utilizado para guardar informação de cada aplicação de forma encriptada, suportando também a existência de utilizadores secundários, i.e., `/data/user_de/${user_id}`[1]. Podemos ver a estrutura do diretorio "`/data/user_de/0/ee.mtakso.client`", através da figura 27.

```
┌──(luis㉿luis)-[~/…/data/user_de/0/ee.mtakso.client]
└─$ tree -d -L 2
.
├── cache
├── code_cache
├── databases
├── files
├── no_backup
└── shared_prefs
```

Figura 27: Espaço privado encriptado de "/data/user_de/0/ee.mtakso.client"

No dispositivo empregue na Tabela 22, página 115, a pasta "`user`"tem 48 subpastas, para um total de 5727 ficheiros, decorrentes do uso da aplicação. Devido à grande quantidade de dados foi necessário consultar de uma forma faseada esses ficheiros para determinar o respetivo valor forense. Seguidamente, procede-se a uma análise aos elementos mais relevantes do ponto de vista forense.

6.2.3 *Análise ficheiros de cache*

Do ponto de vista de uma análise digital forense, o principal elemento da subpasta "`cache`" é o ficheiro `cache/reports_logs.zip` que descompactado corresponde ao

1 Storage Classes, AOSP - Disponível em: https://source.android.com/docs/security/features/encryption/file-based?hl=en#storage-classes - Acedido a 11 Dez. 2023

Listagem 22: Estrutura de diretórios e conteúdo de "/data/user/0/ee.mtakso.client"

```
/data/user/0/
└── ee.mtakso.client/
    ├── app_textures
    ├── app_webview
    │   └── Default
    │       ├── blob_storage
    │       ├── Local Storage
    │       └── Session Storage
    ├── cache
    │   ├── com.google.android.gms.maps.volley
    │   ├── instrument
    │   ├── lottie_network_cache
    │   ├── oat
    │   │   └── arm64
    │   └── WebView
    │       ├── Crashpad
    │       ├── Crash Reports
    │       ├── Default
    │       └── SafeBrowsing
    ├── code_cache
    ├── databases
    ├── files
    │   ├── AFRequestCache
    │   ├── phenotype
    │   │   └── shared
    │   ├── sinch
    │   ├── db
    │   └── splitcompat
    │       └── 2086
    ├── no_backup
    └── shared_prefs
```

ficheiro `report_logs.txt`. Como o nome sugere, este ficheiro armazena mensagens de *logs*. Cada mensagem está rotulada com a indicação temporal (data e hora). As mensagens com maior relevância no contexto forense contêm variada informação sobre o utilizador, tais como a sua localização (ponto de recolha) e o destino que foi definido pelo utilizador. Outros dados relevantes são o *session id*, *user id*, e o tipo

de pagamento utilizado. Como iremos verificar mais adiante, parte do conteúdo do *log* é guardado nas bases de dados locais, sendo posteriormente enviado para os servidores da *Bolt* como parte dos relatórios de analítica.

A Tabela 17 apresenta um resumo do conteúdo para cada um dos restantes ficheiros que fazem parte do diretório de `cache`.

Tabela 17: Sumário da análise ao conteúdo de /data/user/0/ee.mtakso.client/cache

Diretório/ Ficheiro	**Conteúdo**
`/com.google.android.gms.maps.volley`	Dados com compressão de *cache* dos pedidos de rede para troca de informação entre *Google Maps API* e a biblioteca *Volley*. Este é usado para reduzir e acelerar o processo de obtenção de dados relacionados com o Mapa.
`/oat`	Contém os ficheiros com a extensão `".odex"`, que são uma versão compilada da aplicação em formato DEX. Estes são utilizados para melhorar o desempenho das aplicações *Android* ao permitir a sua execução diretamente pelo processador do dispositivo, ao invés de recorrer à máquina virtual *Dalvik*.
`/lottienetworkcache`	Ficheiros para desenho de animações utilizados pela biblioteca *Lottie*.
`/instrument`	- -
`/WebView`	- -
`/report_logs.zip`	Mensagens de *logs* agregadas para criação de relatório de erros/bugs.
journal	Referências para posições de memória de ficheiros de *cache*, ao qual mantém também o seu estado (*Dirty/Clean*)
/	Cópias de pedidos (*GET*) *HTTP*, os pedidos aqui presentes são cópias dos resultados obtidos.

6.2.4 *Bases de dados*

METODOLOGIA. Para facilitar a análise às BD locais, foi empregue o utilitário *sqlite3-android*, que possibilita a leitura das bases de dados SQLite3 presentes no dispositivo móvel, sem que seja necessário a cópia das mesmas para o computador do analista forense. A ferramenta de análise *Objection* permitiu também auxiliar nesta tarefa, no entanto esta opta por criar uma cópia temporária da BD local para a pasta `"/tmp"`da máquina do analista, o que garante a integridade dos dados da aplicação.

Na pasta `databases` existem cinco bases de dados *SQLite3*. Além destas, existe ainda a base de dados `androidx.work.workdb` na pasta `/no_backup`, duas bases de dados na pasta `/app_webview/Default/`, e por fim, a base de dados `main.sqlite` na pasta `/files/sinch/db/950A17B1/`. A Tabela 18 resume as funcionalidades das bases de dados *SQLite3* associadas à aplicação *Bolt*.

CONFIGURAÇÃO PRAGMA. Na perspetiva forense é importante observar as configurações *Pragma* das bases de dados *SQLite3* da aplicação *Bolt*. As definições

Tabela 18: Sumário da análise global das bases de dados em uso pela aplicação

Nome do ficheiro	Descrição
com.google.android. datatransport. events	Dados de analítica e *crash reports* da aplicação a serem enviados para *Firebase console*
google__app__ measurement __local.db	Utilizada para analítica por parte do *SDK Firebase*.
mixpanel	Dados de analítica da interação do utilizador com a aplicação.
clevertap	Dados sobre a aplicação e dispositivo móvel.
chat__db	Mensagens trocadas entre passageiro e condutor.
androidx.work .workdb	Calendarização de tarefas a serem desempenhadas em *background*, gerido pela biblioteca *WorkManager*.
Web Data	Dados para preenchimento automático de informação do utilizador em navegação web.
Cookies	Medidas de segurança de identificação de *bots* e *token* de sessão.
main.sqlite	Configurações e definições para comunicações em tempo real. (As definições guardadas são para o FCM)

mais relevantes são o *Auto Vaccum* e o *Secure Delete*, cujos valores podem ser encontrados através da ferramenta de edição/visualização de bases de dados *SQLite3*, como é o caso do *DB Browser*[2] empregue na análise aqui descrita.

O campo `"Auto Vaccum"`[3] determina a gestão do espaço em disco da base de dados, podendo assumir um de três valores: *none, full* ou *incremental*. As bases de dados da aplicação *Bolt* estão configuradas com o valor full. Esta configuração liberta espaço de forma automática cada vez que dados são removidos da base de dados. No entanto, esta opção não desfragmenta/reorganiza, nem compacta as páginas resultantes da operação. Por sua vez, a configuração `"Secure Delete"`está ativa nas bases de dados, levando a que os registos eliminados sejam reescritos com o *zero byte*, i.e., com o octeto zero, efetivamente removendo fisicamente os dados do suporte persistente. Esta operação é efetuada cada vez que são realizadas operações de remoção ou atualização, não deixando rasto dos valores alterados/removidos, o que nega a possibilidade de um analista forense obter o conteúdo dos valores apagados.

Seguidamente, procede-se à análise detalhada de cada uma das bases de dados cujo conteúdo se considera relevante para uma análise digital forense.

BASE DE DADOS COM.GOOGLE.ANDROID.DATATRANSPORT.EVENTS . Esta base de dados recolhe informação relacionada com eventos que causem falhas

2 DB Browser - Disponível em: https://sqlitebrowser.org/ - Acedido a 12 de Jan. 2023
3 Auto-Vacuum - Disponível em: https://www.sqlite.org/pragma.html#pragma_auto_vacuum - Acedido a 7 de Jul. 2023

da aplicação, recorrendo ao *Firebase Crashlytics (Crashlytics Android SDK/18.2.11)*. A base de dados é composta pelas seguintes sete tabelas: 1) `android_metadata`; 2) `event_metadata`; 3) `event_payloads`; 4) `events`; 5) `global_log_event_state`; 6) `log_event_dropped`; e 7) `transport_contexts`. Na perspetiva forense, as mais relevantes são `event_metadata`, `event_payloads` e `transport_contexts`.

TABELA 'EVENT_PAYLOADS'. Esta tabela é a mais rica em artefactos forenses. Como o nome sugere, a tabela está associada a eventos e ao conteúdo dos mesmos. Cada evento é identificado com um valor numérico guardado no campo `event_id`. O campo *bytes* é do tipo Binary Large OBject (BLOB) e agrega, em formato JSON, os dados empregues pelo *crashlytics*. Conforme ilustra a Listagem 23, a tabela guarda identificadores associados ao dispositivo e ao utilizador. O campo '`startedAt/endedAT`' corresponde à data/hora em que foi iniciado/terminado o processo de criação do relatório de erro. O formato empregue é o *Unix Epoch UTC*, com precisão do segundo. Por exemplo, o valor 1673382240 corresponde à data *Tue 10 January 2023 20:24:00 UTC*. O campo "*jailbroken*"com o valor "*false*" indica, de forma incorreta no caso deste trabalho, que a aplicação está a ser executada num dispositivo regular, i.e., sem *root*. A recolha de informação referente ao nível da bateria é uma informação que pode também ser considerada excessiva, eventualmente tolerada com o intuito que o GPS possa ser desligado para poupança de bateria, o que pode colocar a localização em risco. Note-se que é dito que a aplicação Uber faz uso do nível de bateria do dispositivo como um dos parâmetros que influencia o algoritmo dinâmico de fixação do preço do serviço, com os utilizadores cujo dispositivo apresentem pouca carga, a serem alvos de um preço maior, dado que é expectável que não queiram arriscar ficar sem carga[4]. Não obstante, segundo as informações presentes no *website* oficial da *Bolt*[5], não existe referência ao aumento de preço associado ao nível da bateria.

Além dos dados já mencionados,como seria de esperar, cada um dos relatórios descreve em detalhe as funções que estavam a ser executadas quando o erro ocorreu. Existe também um separador de atributos específicos, no formato de chave-valor (*key-values*), o qual permite no mesmo relatório de erros seja utilizado para as restantes aplicações da *Bolt*, tais como o *Bolt Food Courier, Bolt Food* e *Bolt Driver*. Podemos assim inferir que as restantes aplicações da *Bolt* utilizam esta mesma

4 Users Are More Likely To Pay Surge Pricing If Their Phone Battery Is Low, Forbes - Disponível em:https://www.forbes.com/sites/amitchowdhry/2016/05/25/uber-low-battery/ - Acedido a 15 Dez. 2023

5 Price higher than expected, Bolt - Disponível em:https://bolt.eu/en/support/articles/115002906513/ - Acedido a 15 Dez. 2023

Listagem 23: (Excerto) Tabela `"event_payloads"`, campo `"bytes"`, referência interna do utilizador e dispositivo


```
        "---omitido para abreviação---":...,
        "session": {
                "generator": "Crashlytics Android SDK/18.2.11",
                "identifier": "NjNCRE...valor.omitido...AyRTY2QTE=",
                "startedAt": 1673382240,
                "endedAt": 1673385994,
                "crashed": false,
                "app": {
                        "identifier": "ee.mtakso.client",
                        "version": "2086",
                        "displayVersion": "CA.52.1",
                        "installationUuid": "9315b...valor.omitido...ebd55"
                },
                "user": {
                        "identifier": "14...valor.omitido...00"
                },
                "os": {
                        "platform": 3,
                        "version": "9",
                        "buildVersion": "REL",
                        "jailbroken": false
                },
                "device": {
                        "arch": 9,
                        "model": "Redmi Note 3",
                        "cores": 6,
                        "ram": 2976002048,
                        "diskSpace": 26283376640,
                        "simulator": false,
                        "state": 0,
                        "manufacturer": "Xiaomi",
                        "modelClass": "aosp_kenzo"
                },
                "---omitido para abreviação---":...,
                "device": {
                        "batteryLevel": 0.9700000286102295,
                        "batteryVelocity": 2,
                        "proximityOn": true,
                        "orientation": 1,
                        "ramUsed": 1817567232,
                        "diskUsed": 5343428608
                },
                "---omitido para abreviação---":...,
```


estrutura de relatório de erros para realizar o envio de erros registados. O que permite estender os resultados aqui descritos para as restantes aplicações *Bolt*, respeitantes à comunicação de erros encontrados.

Uma das partes mais relevantes para análise forense digital, é a secção designada por `"log"`. Esta contém muita da informação que foi apresentada ao utilizador pela aplicação, além da parte afeta às transações monetárias – saldo disponível, valor pago, método de pagamento –, bem como a localização de partida e destino, incluindo morada e as respectivas coordenadas de GPS.

Na Listagem 24 podemos observar que conjuntamente com as mensagens de log, são enviadas as localizações GPS do destino e ponto de partida quando se requisita

uma viagem. Além desses dados, é também recolhida a informação empregue para depuração de rede, com a adição das mensagens de POST que foram enviadas. Novamente, esta informação referente ao ponto de recolha – destino, método de pagamento – é mais pormenorizada do que aquela que é apresentada pela interface da aplicação ao utilizador.

Listagem 24: (Excerto) Tabela **"event_payloads"**, campo **"bytes"**, secção **log** - Informação de localização do utilizador

```
"---omitido para abreviação---":...,
"log": {
        \n3634701 RideHailing - PreOrder: Requesting new transaction with
        ↪  parameters = InternalResult(pickup=39.632062,-8.831134,
        ↪  destinations=Destinations(items=[Destination(destinationPlace=Estrada
        ↪  Principal ...valor.omitido...,
        ↪  39...valor.omitido...,-8...valor.omitido..., isVisited=false,
        ↪  smartPickup=null, smartPickupArea=null)]),
"---omitido para abreviação---":...,
        paymentInformation=PaymentInformation(selectedBillingProfile=PersonalPro
        ↪  file() BillingProfile(id='null', name='Pessoal',
        ↪  templateName='personal', isSelected=true, isEditable=false,
        ↪  isDeletable=false,
        ↪  selectedPaymentMethod=PaymentMethod(id=M49J...valor.omitido...Z43,
        ↪  type=adyen, iconUrl=https://static.bolt.eu/payment/icons/visa.png,
        ↪  name=···· 8759,
"---omitido para abreviação---":...,
        \n3728780 network call: --> 71b19f49-11f1-4179-ba95-2dde6e4f16be \tURL:
        ↪  POST https://user.live.boltsvc.net/mobility/search/poll?version=CA.5
        ↪  2.1&deviceId=fQ4...valor.omitido...yx15&device_name=XiaomiRedmi%20No
        ↪  te%203&device_os_version=9&channel=googleplay&deviceType=android&cou
        ↪  ntry=pt&language=pt-PT&gps_lat=39...valor.omitido...62&gps_lng=-8...
        ↪  valor.omitido...38&user_id=14...valor.omitido...&session_id=140755..
        ↪  .valor.omitido...&rh_session_id=140755...valor.omitido...
"---omitido para abreviação---":...,
        Cookie: __cf_bm=dsWQ6W1DgzMjTkc3JtJFIkxB800RgmYCADZX.QcdWJA-1673384822-0
        ↪  -AfoX2tg2v+sa3bJRjezmqYiHipypMvsPNvgNCdL2pvGCT/REXlk/eWB0/EkX/Q3bsxhR
        ↪  mtXPWphxVE1lE0kckJs= ||| User-Agent: okhttp/4.9.1 \tBody:
        ↪  {\"destination_stops\":[{\"lat\":39...valor.omitido...,\"lng\":-8...
        ↪  valor.omitido...}],\"payment_method\":{\"id\":\"M49J...valor.omitido
        ↪  ...\",\"type\":\"adyen\"},\"pickup_stop\":{\"lat\":39...valor.omitid
        ↪  o...,\"lng\":-8...valor.omitido...},\"stage\":\"category_selection\"
        ↪  ,\"viewport\":{\"north_east\":{\"lat\":39...valor.omitido...,\"lng\"
        ↪  :-8...valor.omitido...},\"south_west\":{\"lat\":39...valor.omitido..
        ↪  .,\"lng\":-8...valor.omitido...}}} \t--> END POST  (332-byte body).
        ↪  \n3733009
"---omitido para abreviação---":...,
        bolt_subscription_enabled=false, user_lat=39...valor.omitido...,
        ↪  user_lng=-8...valor.omitido..., type of
        ↪  exception=SocketTimeoutException, user_id=14...valor.omitido...,
        ↪  service_desk_report=false, rh_session_id=140755200u1673385859,
        ↪  location_kit=google, screen_name=Confirm Category, type=action,
        ↪  device_id=fQ4Hf...valor.omitido...,
"---omitido para abreviação---":...,
```

É ainda mantida informação relativa ao saldo e valor da(s) viagem(ns), conforme ilustra a Listagem 25(página 121), a qual foi retirada dos *logs* da aplicação. Esta informação é também considerada pessoal e sensível, parecendo excessiva. De facto, tais dados não aparentam ser necessários para funções de depuração, assim como as opções para indicação da existência ou não de promoções aplicadas ao valor, do

preço por km, das taxas de cancelamento, entre outros elementos. É nossa opinião que tal recolha de dados sensíveis é demasiadamente abrangente para as necessidades dos programadores da aplicação, levantando-se a dúvida se mantém o pleno respeito pelo Regulamento Geral sobre a Proteção de Dados (RGPD).

Listagem 25: (Excerto) Tabela `event_payloads"`, campo "bytes- Informação de pagamento e saldo, demasiada informação de *log*

```
balanceInformation=Optional.of(Balance(title=Saldo Bolt,
↪  balanceItems=[BalanceItem(amountHtml=<font color=\"#EB4755\">-6.79€</font>,
↪  isMain=true)], topUpLink=BalanceLink(titleHtml=Adicionar dinheiro, url=https
↪  ://balance.bolt.eu/packages?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

"---omitido para abreviação---":...,

price=RideOptionsCategoryPrice(primaryHtml=<font
↪  name='body_semibold_1'>3.75€</font>, secondaryHtml=null,
↪  lockHash=4ZS87hjvtvvABp7R, promoString=null, promoApplied=null,
↪  surgeString=null, actualString=3.75€),
↪  action=Order(showSurgeConfirmation=false),
↪  info=[RideOptionsCategoryInfoGroup(name=Tarifa, value=3.75€,
↪  subgroups=[RideOptionsCategoryInfoSubGroup(name=Preço mínimo, value=3.75€),
↪  RideOptionsCategoryInfoSubGroup(name=Começa, value=1.06€),
↪  RideOptionsCategoryInfoSubGroup(name=Por km, value=0.69€/KM),
↪  RideOptionsCategoryInfoSubGroup(name=Por min, value=0.10€/MIN)]),
↪  RideOptionsCategoryInfoGroup(name=Tempo de espera, value=0.25€/MIN,
↪  subgroups=null), RideOptionsCategoryInfoGroup(name=Taxa de cancelamento,
↪  value=3.75€, subgroups=null), RideOptionsCategoryInfoGroup(name=Lugares,
↪  value=4, subgroups=null)]
"---omitido para abreviação---":...,
```

TABELA 'EVENT_METADATA'. Esta tabela contém informação relevante para *crash reports*, nomeadamente uma descrição das características do dispositivo móvel, tais como o *sdk version*, a OS *build*, versão do *firmware, app build*, fabricante, entre outros itens. Ficamos assim a conhecer a informação recolhida para ser enviada em conjunto com a restante analítica presentes nas restantes tabelas desta base de dados.

BASE DE DADOS 'CHAT.DB'. Esta base de dados pode ser bastante relevante para uma análise forense digital, uma vez que armazena as mensagens escritas trocadas entre o passageiro e o condutor da viatura de transporte. A troca de mensagens apenas pode ocorrer após o pagamento do serviço solicitado, o que significa que já existe um condutor atribuído ao serviço solicitado. O sistema envia de forma automática mensagens de confirmação com os dados do condutor e viatura ao passageiro, o que leva a que sejam criadas mensagens para todas as viagens efetuadas, independentemente de ter existido ou não troca direta de mensagens entre

o condutor e o passageiro. A base de dados chat.db tem as seguintes cinco tabelas: 1) `android_metadata`; 2) `chat`; 3) `chat_messages`; 4) `chat_terminal_messages`; e 5) `room_master_table`. As tabelas mais relevantes do ponto de vista de análise digital forense são `chat`, `chat_messages` e `chat_terminal_messages`. As mensagens associadas a uma determinada viagem são marcadas através de um identificador único, apropriadamente designado por "`id`", empregue em todas as tabelas associadas ao serviço de *chat*. Seguidamente, analisam-se as tabelas mais relevantes para uma análise forense.

TABELA 'CHAT'. Os campos da tabela *chat* encontram-se sumariamente descritos na Tabela 19 (página 122). O campo `id` identifica a troca de mensagens, e serve de chave forasteira nas restantes tabelas associadas ao sistema de troca de mensagens. O campo `start_date` representa a data/hora, em formato *Unix timestamp* milissegundos, da criação da mensagem, um elemento valioso numa análise forense. Os outros campos da tabela identificam o condutor, a viatura e a viagem.

Tabela 19: Descrição sumária dos campos da tabela `chat`

Campo	**Tipo dados**	**Descrição**
`id`	`text`	Identificador único para troca de mensagens
`title`	`text`	Primeiro nome do condutor
`description`	`text`	Descrição da viatura (Marca/Modelo e Matrícula)
`thumbnail_url`	`text`	*Url* para foto do condutor
`start_date`	`integer`	Data da criação da mensagem em Unix (milisegundos)
`order_handle_order_id`	`integer`	Identificador único associado ao pedido da viagem
`order_handle_order_system`	`text`	Dependendo do tipo de pedido (*taxi, deliver*)
`order_handle_city_id`	`integer`	Identificador único associado à cidade onde o pedido foi efetuado.

TABELA CHAT_MESSAGES. Os campos mais relevantes da tabela `chat_messages` são sucintamente descritos na Tabela 20. Esta tabela armazena as mensagens trocadas entre ambos os intervenientes da viagem. Conforme referido anteriormente, o campo `chat_id` corresponde ao campo `id` da tabela `chat`, e atua na tabela `chatmessages` como chave forasteira, identificando as mensagens trocadas no

contexto de uma viagem *Bolt*. O campo status permite identificar os diferentes estados que a mensagem pode apresentar, o qual pode ser um contributo bastante útil para perceber se o passageiro consultou a mensagem, ou se a mesma não chegou sequer a ser entregue.

Tabela 20: Descrição sumária dos campos da tabela `chat_messages`

Campo	**Tipo dados**	**Descrição**
`id`	`text`	Identificador único para cada mensagem enviada.
`chat_id`	`text`	Chave estrangeira que permite relacionar com o condutor e veículo da tabela `chat`.
`text`	`text`	Cópia das mensagens trocadas entre cliente e condutor.
`sender_id`	`text`	*Tag "rider/driver"*, seguido do identificador do autor da mensagem (Exemplo:`"rider:140755200"`).
`sender_name`	`text`	Primeiro nome do autor da mensagem.
`quick_reply_id`	`text`	Identificador do autor da mensagem de resposta.
`date`	`integer`	Data da criação da mensagem em Unix (milisegundos).
`status`	`integer`	*1 = sending error, 2 = sending, 3 = preliminary delivered to backend, 4 = delivered to backend, 5 = delivered to recipient, 6 = local seen by recipient, 7 = seen by recipient.*

TABELA 'CHAT_TERMINAL_MESSAGES'. Na Tabela 21 encontram-se sumariamente descritos os campos mais importantes, do ponto de vista forense, da tabela `chat_terminal_messages`. A estrutura da tabela é similar à encontrada nas restantes tabelas da base de dados `chat.db`. Assim, existe igualmente o campo `chat_id`, atuando novamente como chave forasteira do id identificador do conjunto de mensagens trocadas no âmbito de um viagem (ou pedido de viagem). O campo status regista se a viagem foi efetivamente realizada (*finished*), ou se pelo contrário foi cancelada (*aborted*). A informação referente ao estado da viagem também é registada através da string guardada no message. Finalmente, o campo date, como o nome indica, representa a data/hora de chegada, em formato *Unix timestamp* milissegundos de uma viagem, caso a mesma se tenha concretizado.

BASE DE DADOS 'CLEVERTAP'. Esta base de dados não apresenta no seu conteúdo informação relevante num contexto forense. Foram analisadas todas as tabelas desta base de dados, por forma a garantir que os seus campos não continham informação que fosse considerada como sensível. Os campos da única tabela que poderia conter informações pessoais (Tabela `user_Profiles`), desta base de dados,

Tabela 21: Descrição sumária dos campos mais relevantes da tabela chat_terminal_messages

Campo	Tipo dados	Descrição
`id`	`text`	Identificador único para cada mensagem de término de viagem.
`chat_id`	`text`	Chave estrangeira que permite relacionar o condutor e o seu veículo da tabela `chat`.
`message`	`text`	Mensagem que confirma o término da viagem.
`status`	`text`	A viagem pode também ser cancelada além de terminada o que origina a um diferente estado (*aborted, finished*).
`date`	`integer`	Data da criação da mensagem em Unix (milisegundos).

encontram-se "vazios". Os campos aos quais nos referimos como "vazios"são os campos: `user_last_name`, `user_first_name`, `user_phone`, da Tabela `userProfiles` da base de dados `clevertap`, o resultado encontrado é apresentado na Listagem 26.

Listagem 26: Tabela *"userProfiles"*, campo *"Data"*


```
{
        "country": "pt",
        "user_email": "",
        "storage_cache_size": 58822656,
        "app_version": "CA.52.1",
        "user_last_name": "",
        "rh_installation_source": "side-loaded",
        "language": "pt-pt",
        "rentals": "false",
        "rh_accessibility_enabled": false,
        "storage_data_size": 66490368,
        "platform": "android",
        "food": "false",
        "device_name": "XiaomiRedmi Note 3",
        "user_first_name": "",
        "push_enabled": true,
        "platform_version": "9",
        "user_id": 140755200,
        "facebook_connected": false,
        "device_was_logged_in": true,
        "user_phone": "",
        "device_total_free_size": 20417282048,
        "drive": "false",
        "storage_total_size": 125313024,
        "ridehailing": "true"
}
```

6.2.5 *Análise shared preferences*

O diretório `shared_prefs` contém diversos ficheiros destinados a guardar definições da aplicação, ou dados específicos de reduzidas dimensões através de pares *key-values*. As duas formas mais comuns para guardar este tipo de dados é através da `SharedPreferences`[6] disponível desde a primeira versão *Android*, e o `Preferences DataStore` que surgiu desde o *Android* 11 (API 30).

Na programação, o objeto *SharedPreferences* pode ser declarado de forma global o que pode originar inadvertidamente que qualquer aplicação possa ler o seu conteúdo, e consequentemente a exposição de dados sensíveis caso os mesmos estejam a ser guardados nestes ficheiros. Por sua vez o *Preferences DataStore*[7], é uma implementação do *DataStore* foi criado para substituir o *SharedPreferences*. O *Preferences DataStore* apresenta várias vantagens em relação ao seu antecessor, desde a manipulação mais eficiente de dados, melhor desempenho ao efetuar as suas tarefas de forma assíncrona, especialmente em cenários com grandes volumes de dados.

Na versão *Bolt CA*.52.1 existem 41 ficheiros XML no diretório `shared_prefs`. Dado o elevado número de ficheiros XML, foi criada a Tabela 22, que procura de uma forma sucinta descrever os ficheiros mais relevantes para uma análise forense.

Tabela 22: Ficheiros XML mais relevantes

Nome ficheiro	**Valor forense**
`com.google.android.gms.appid`	*Token* de autenticação de cliente (Para comunicação com a Google API), *timestamp* e versão da aplicação.
`com.google.android.gms.measurement.prefs.xml`	Campos relevantes: `first_open_time,health_monitor:start,previous_os_version,last_pause_time`.
`FirebaseHeartBeat`	Informação que identifica o dispositivo móvel *(device name, device brand, device model).*
`com.google.maps.api.android.lib6.drd.PREFERENCES_FILE`	Identifica o País através do campo *LegalCountry.*

Na análise dos ficheiros existentes no diretório `shared_prefs` foi possível determinar que nestes não se encontrava qualquer informação sensível do utilizador que não estivesse protegida por encriptação. Assim, a versão analisada não expõe informação sensível do utilizador nos ficheiros do diretório `shared_prefs`. Os dados cifrados obviamente requerem o conhecimento do algoritmo e da chave de encriptação para

6 API SharedPreferences: https://developer.android.com/training/data-storage/shared-preferences - Acedido a 31 de Jan. 2023

7 DataStore, Google Developers - Disponível em: https://developer.android.com/topic/libraries/architecture/datastore?hl=pt-br - acedido a 21 Dez. 2023

ser possível decifrar o respetivo conteúdo. Os identificadores utilizados pelas diversas bibliotecas de analítica utilizam identificadores de forma a manter o anonimato do utilizador. Ou seja, este relacionamento é feito através de identificadores únicos, tais como `deviceId,installation.id`, ou `accountId`.

FICHEIRO: 'COM.GOOGLE.ANDROID.GMS.MEASUREMENT.PREFS.XML'. Este ficheiro contém informação temporal de valor forense. Esta informação é empregue para acesso às API's da *Google*, neste caso específico do *Google Maps*. A API do *Google Maps* é utilizada para efeitos de localização do utilizador, bem como a demonstração de rota, etc. Deste modo, a informação temporal existente no ficheiro pode ser usada para contextualizar no tempo a interação do utilizador com o mapa. De facto, quando se inicia a aplicação *Bolt*, é de imediato apresentado o mapa. Assim, o campo `first_open_time` regista a data/hora – *timestamp* em *Unix* milissegundos – correspondente à abertura da aplicação pela última vez. Através do campo `previous_os_version` podemos verificar se a versão do dispositivo foi alterada desde a última utilização, o que pode indicar que o utilizador fez uso de outro dispositivo para aceder à conta. Similarmente, o campo `has_been_opened` indica se a aplicação já foi aberta anteriormente neste dispositivo, o que novamente pode ajudar a perceber se a aplicação foi utilizada anteriormente. O campo `last_pause_time` é um campo de data/hora, o qual regista a última vez em que a aplicação pausou a execução, que tal como os demais campos deste género em base de dados da aplicação *Bolt*, está em formato *Unix timestamp* milissegundos.

FICHEIRO: 'EE.MTAKSO.CLIENTINSTALLATION__PREFERENCES__ __ANDROIDX__SECURITY__CRYPTO__ENCRYPTED__PREFS.XML' Os programadores da aplicação optaram por cifrar as preferências de instalação que são guardadas pela aplicação. Assim, o acesso a credenciais de autenticação ou a outra informação privada está protegido. A Tabela 23 lista os ficheiros XML cifrados existentes na pasta `shared_prefs`.

O nome dos ficheiros indica que os mesmos foram cifrados com recurso à biblioteca *AndroidX Security Crypto*. A API `androidx.security.crypto` permite gerir chaves, cifrar ficheiros e manter a segurança das *shared preferences*[8].

Este capítulo destacou os artefactos forenses mais relevantes que foram encontrados, gerados pela aplicação decorrentes da interação do utilizador com a aplicação.

8 AndroidX Security: https://developer.android.com/jetpack/androidx/releases/security - Acedido a 13 Fev. 2023

Tabela 23: Ficheiros XML cifrados em `shared_prefs`

Nome dos ficheiros cifrados
`ee.mtakso.client_config_preferences__androidx_security_crypto_encrypted_prefs.xml`
`app_validation_status__androidx_security_crypto_encrypted_prefs.xml`
`carsharing_content_display_configs__androidx_security_crypto_encrypted_prefs_iv.xml`
`carsharing_forever_dismissed_banners__androidx_security_crypto_encrypted_prefs_iv.xml`

Terminada análise estática e identificados os artefactos forenses mais relevantes, no próximo capítulo irá ser executada a análise dinâmica da aplicação.

7

ANÁLISE DINÂMICA

Independentemente das técnicas de ofuscação utilizadas para dificultar a tarefa de análise estática em aplicações móveis, não existe forma das aplicações evitar a sua instrumentalização. Isto acontece dado que uma aplicação é executada dentro do espaço de sistema conhecido por *Android framework*. Uma vez que controlamos o ambiente em que as mesmas são executadas é possível verificar a lógica por detrás da sua execução.

Para realizar a análise dinâmica é necessário começar por explorar todas as funcionalidades da aplicação pelo menos uma vez.

7.1 OWASP - MOBILE APPLICATION SECURITY (MAS)

É importante referir o contributo para este trabalho da *OWASP*. De facto, com o seu projeto Mobile Application Security (MAS)[1], a *OWASP* definiu uma norma de segurança para dispositivos móveis, composto pelo Mobile Application Security Verification Standard (MASVS) e o Mobile Application Security Testing Guide (MASTG).

O *MASVS* foi criado com o intuito de ser uma norma que estabeleça um nível de confiança no desenvolvimento seguro de aplicações móveis, podendo ser empregue das seguintes formas:

- Usar como uma métrica - Para providenciar um padrão de segurança com o qual as aplicações móveis podem ser comparadas pelos programadores e proprietários das aplicações;
- Usar como um guia - Para orientar durante todas as fases de desenvolvimento e teste das aplicações móveis;
- Usar durante concepção - Para fornecer um ponto de partida para a verificação da segurança das aplicações móveis.

1 OWASP Mobile Application Security, Disponível em: https://mas.owasp.org/ - acedido a 10 Janeiro de 2024

É importante notar que o MASVS apenas cobre a segurança da aplicação móvel (lado do cliente) e da comunicação de rede entre a aplicação e os seus pontos de acesso remotos, assim como alguns requisitos básicos e genéricos relacionados com autenticação do utilizador e gestão de sessão. Não inclui requisitos específicos para serviços remotos (ex: *web services*) associados à aplicação para além de um conjunto limitado de requisitos genéricos relacionados com autorização, autenticação, verificação de controlo e gestão de sessão.

Por sua vez o MASTG é um manual para testar a segurança de aplicações móveis. Descreve os processos técnicos para verificar os requisitos listados em MASVS. O MASTG inclui uma lista de casos de teste, cada um correspondendo a um requisito no MASVS. Enquanto que os requisitos do MASVS são de alto nível e genéricos, o MASTG fornece recomendações extensivas e procedimentos de testes específicos consoante o SO.

Não existindo uma ligação à equipa de desenvolvimento do software desta aplicação, este trabalho utiliza parte da informação que nos é prestada pelo MASTG, para garantir que foram testados os procedimentos de segurança mais relevantes numa aplicação móvel *Android*.

7.1.1 *Verificar quanto à existência de informação sensível nos Logs*

Para verificar os *logs* produzidos pela aplicação em tempo real foi utilizada a ferramenta *Logcat*. Também na análise dinâmica, o *Logcat* é uma ferramenta importante que nos ajuda a perceber se existe informação sensível a ser escrita para os ficheiros de *log*.

Após correr a aplicação é possível verificar todos os *logs* que estão associados a um processo, sendo possível monitorizar o comportamento da aplicação.

O comando apresentado na listagem 27 permite consultar os *logs* do processo correspondente ao *package name* da aplicação. Conseguimos desta forma obter um maior volume de informação com a referência do processo, em relação ao uso apenas do *package name*.

Listagem 27: Exemplo: Obter o `PID` da aplicação *Bolt* e consultar *Logs*

```
$ adb shell ps | grep -i ee.mtakso
u0_a124   13025 449   1989096 348604 ffffffff 00000000 S ee.mtakso.client

$ adb logcat ps | grep -i 13025
```

O resultado do comando apresentado na listagem 27 revela que existe muita informação a ser escrita para os *logs*, cabendo ao analista identificar se a informação é realmente privada e/ou sensível.

Após analisar o conteúdo apresentado pelo *logcat*, foi criada a lista que se apresenta de seguida com os pontos mais relevantes que foram encontrados.

Análise *logcat*:

- Uso extensivo de mensagens de *debugging*;
- Grosso das mensagens de *debugging* são enviadas para *Firebase Analytics*;
- Envio de *logs* para a Google Cloud Console (GCC);
- Escrita de informação sobre o *Wi-Fi state*;

O uso extensivo de mensagens de *log* por si só não é prejudicial, no entanto mensagens de *stack strace* podem revelar informação sobre a lógica da aplicação. Outro problema que advém do uso excessivo de informação escrita para *logs*, prende-se com a diminuição de desempenho no dispositivo, o que por sua vez irá também afetar a quantidade de informação que é enviada para uma localização remota. Além da grande dimensão dos *logs* recolhidos pela aplicação *Bolt*, acresce-se o facto de parte desta informação ser guardada localmente, como foi referido na análise estática, sendo a mesma posteriormente enviada para servidores remotos de acordo com os *trackers* existentes.

7.1.2 *Permissões*

Uma forma de confirmar a correta utilização das permissões requeridas no `Manifest`, após a aceitação por parte do utilizador é através da interação com a aplicação. A listagem 28, obtida através do ADB, apresenta as permissões requeridas que foram aceites pelo utilizador, estando de acordo com as permissões definidas pelo `Manifest`, conforme discutido anteriormente na Secção 3.8.2.

Observa-se que as permissões aceites pelo utilizador correspondem às permissões declaradas no `Manifest`, sendo somente empregues aquelas que foram aceites no decorrer do uso da aplicação.

Foram discutidas em detalhe as permissões encontradas no `AndroidManifest.xml` e a sua relevância para a aplicação, as quais podem ser consultadas na Secção 3.8.2.

Listagem 28: Obter a lista de permissões utilizadas através de adb e dumpsys

```
$ adb shell dumpsys package ee.mtakso.client | grep permission

declared permissions:
requested permissions:
android.permission.INTERNET
... <omitido conteudo não relevante para a demonstracao> ...
install permissions:
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE:
↪  granted=true
com.google.android.c2dm.permission.RECEIVE: granted=true
android.permission.MODIFY_AUDIO_SETTINGS: granted=true
android.permission.NFC: granted=true
android.permission.FOREGROUND_SERVICE: granted=true
android.permission.RECEIVE_BOOT_COMPLETED: granted=true
android.permission.BLUETOOTH: granted=true
android.permission.INTERNET: granted=true
android.permission.GET_PACKAGE_SIZE: granted=true
android.permission.ACCESS_NETWORK_STATE: granted=true
android.permission.VIBRATE: granted=true
android.permission.ACCESS_WIFI_STATE: granted=true
com.google.android.gms.permission.AD_ID: granted=true
android.permission.WAKE_LOCK: granted=true
runtime permissions:
android.permission.ACCESS_FINE_LOCATION: granted=true
android.permission.ACCESS_COARSE_LOCATION: granted=true
```

Importa notar que se identificou o uso do método `checkSelfPermission` pelas bibliotecas de terceiros empregues na aplicação *Bolt*. Este método verifica se a permissão pretendida se encontra no `Manifest` da aplicação, para validar a operação.

7.1.3 *Análise de tráfego de rede*

A grande maioria das aplicações móveis, assentam num modelo de comunicação cliente-servidor, para validação de informação pessoal sensível, bem como na obtenção de informação de negócio.

Sendo esta informação que transita entre a aplicação móvel DM do cliente e o servidor caracterizada como sensível, é importante perceber se os corretos mecanismos de segurança estão a ser aplicados.

A partir do *Android* 7.0 (SDK igual ou superior a API 24), começou a confiar apenas em certificados de sistema para estabelecer ligações SSL/TLS. Antes da versão Android 7.0, eram aceites ambos os certificação pré-instalados e certificados de sistema. Essa mudança foi feita para melhorar a segurança das ligações SSL/TLS em dispositivos Android.

Ao consultar o `Manifest`, a aplicação aceita um *Android* SDK mínimo como 21, no entanto o *target* do SDK é 32, como tal apenas certificados de sistema são

aceites. Temos que ter isso em conta se queremos analisar o tráfego que transita entre aplicação móvel e servidor. Não obstante a aplicação *Bolt* em estudo ter como requisito mínimo a versão 21 do *Android* SDK, a versão declarada do SDK alvo é a 32. Disto depreende-se que as ligações seguras requerem forçosamente certificados digitais reconhecidos pelo sistema *Android*. Tal dificulçta a captura do tráfego cliente-servidor que se encontra necessariamente cifrado.

Existem três formas de ultrapassar essa restrição:

- Alterar a aplicação para confiar em certificados instalados pelo utilizador;
- Inserir o certificado criado pelo utilizador com os restantes certificados de sistema;
- Manipular a aplicação durante a sua execução;

7.1.3.1 *Protocolos utilizados*

Numa análise de tráfego é importante analisar os protocolos que são utilizados pela aplicação e se estes são ou não encriptados. Para o efeito foram empregues as ferramentas `TcpDump` e `Wireshark`, na qual utilizamos as configurações definidas anteriormente nas secções 4.3.7 e 4.3.8 respetivamente.

Para testar os protocolos utilizados foram despoletadas todas as ações permitidas pela aplicação. Os resultados obtidos baseiam-se no tráfego recolhido `formato Pcap`, sendo resumidos de seguida.

Protocolos utilizados:

- HTTPs;

A Tabela 24 apresenta uma lista de domínios e *cloud providers* associados aos endereços IPv4 encontrados no tráfego capturado entre o DM e um *endpoint* remoto. Por sua vez a tabela 25 apresenta os endereços IPv6 detetados no tráfego PCap.

Esta informação é relevante para perceber se existe comunicação com *endpoints* não confiáveis. Isto porque apesar da comunicação ser encriptada, é importante perceber se a mesma está a ser transmitida a servidores indevidos ou não confiáveis.

No separador estatísticas, sub-menu *conversations* do *Wireshark* é possível observar os portos que foram utilizados para trocar mensagens. Todas as trocas de mensagens TCP e User Datagram Protocol (UDP) dos *endpoints* anteriores ocorreram utilizando o porto 443 à excepção do *endpoint* "2a00:1450:400c:c06::bc", o qual utilizou porto 5228 (tcp) para comunicar. Para determinar o serviço alojado no referido porto 5228/TCP, efetou-se uma análise com recurso à ferramenta Nmap,

Tabela 24: Informações sobre endpoints IPv4 encontrados no PCap - Bolt

Endereços	**Cloud Provider**	**Domínio**	**Organização**
13.225.244.17, 13.225.244.18, 13.225.244.29, 13.225.244.84, 13.225.244.95	Amazon	Cloudfront.net	-
35.190.88.7	Google	Bugsnag.com, Googleusercontent.com	-
52.219.169.107	Amazon	Amazonaws.com	-
64.4.245.84	-	Paypal.com	-
91.212.42.181	-	Ayden.com	-
94.46.175.183	-	B-CDN.net	-
104.18.10.117	-	Cloudfare	-
107.178.240.159, 130.211.34.183	Google	MixPanel.com, Googleusercontent.com	-
142.250.200.138	-	1E100.net	Google LLC
146.75.89.21, 146.75.89.35, 192.229.221.25	-	Paypal.* (múltiplos subdomínios paypal)	-

encontrando-se os resultados na Listagem 29. A segunda parte da Listagem 29 demonstra o uso da ferramenta Masscan que permite correr um varrimento por todas as portas do *endpoint* remoto, expondo os portos que se encontram abertos. Embora não tenha sido possível identificar o serviço associado ao porto 5228/TCP no endpoint em análise, verificou-se que o mesmo recorre a um certificado TLS.

7.2 ANÁLISE BURPSUITE

Ao trabalhar com intercepção de tráfego é sempre necessário recorrer a um software que consiga fazer o papel de *proxy*. A sua necessidade pretende-se tal como o nome nos diz com intercetar o tráfego, e posteriormente no caso de se tratar de tráfego encriptado, a necessidade da sua desencriptação.

Existem diversas ferramentas comerciais e *open-source* para desempenhar este papel, cada uma oferece diferentes funcionalidades, bem como pontos fortes e fracos em áreas distintas. A ferramenta escolhida para desempenhar este papel foi o '*BurpSuite Community Edition*', pelas razões apresentadas na Secção 4.3.9.

Tabela 25: Endpoints IPv6 encontrados no PCap - Bolt

Endereços (Ipv6)		
2001:4860:4802:32::36	2001:4860:4802:34::36	2600:1901:0:7a0b::
2600:9000:21d6:0:1d:e2ad:55c0:93a1	2600:9000:21d6:5a00:1d:e2ad:55c0:93a1	2606:4700::6812:14e3
2606:4700::6812:1f47	2606:4700::6812:a75	2606:4700::6812:b75
2a00:1450:4003:803::2003	2a00:1450:4003:803::2008	2a00:1450:4003:806::2003
2a00:1450:4003:806::2004	2a00:1450:4003:807::2003	2a00:1450:4003:807::200a
2a00:1450:4003:808::200e	2a00:1450:4003:80a::200a	2a00:1450:4003:80c::200a
2a00:1450:4003:80c::200e	2a00:1450:4003:80f::200d	2a00:1450:4003:80f::200e
2a00:1450:4003:811::2004	2a00:1450:4003:811::200a	2a02:26f0:7900::172f:bc69
2a02:26f0:7900:196::f09	2a02:26f0:7900:196::3282	2a02:26f0:7900:1a4::3282
2a00:1450:400c:c06::bc	2a03:2880:f052:11:face:b00c:0:2	

Listagem 29: Nmap: Scan ao porto 5228 do Endereço IPv6 (TCP)

```
1
2  $ nmap -6 -A -p 5228 2a00:1450:400c:c06::bc
3  Starting Nmap 7.94 ( https://nmap.org ) at 2023-08-10 15:13 WEST
4  Nmap scan report for wb-in-f188.1e100.net (2a00:1450:400c:c06::bc)
5  Host is up (0.075s latency).
6
7  PORT     STATE SERVICE VERSION
8  5228/tcp open  ssl/nim IBM AIX Network Installation Management
9  |_ssl-date: TLS randomness does not represent time
10 | ssl-cert: Subject: commonName=invalid2.invalid
11 | Not valid before: 2015-01-01T00:00:00
12 |_Not valid after:  2030-01-01T00:00:00
13 Service Info: OS: AIX; CPE: cpe:/o:ibm:aix
14 ----------------------------------------------------------------------
15 $ sudo masscan -p- 2a00:1450:400c:c06::bc --rate=10000
16 Initiating SYN Stealth Scan
17 Scanning 1 hosts [65535 ports/host]
18 Discovered open port 5230/tcp on 2a00:1450:400c:c06::bc
19 Discovered open port 443/tcp on 2a00:1450:400c:c06::bc
20 Discovered open port 5228/tcp on 2a00:1450:400c:c06::bc
21 Discovered open port 5235/tcp on 2a00:1450:400c:c06::bc
22 Discovered open port 5223/tcp on 2a00:1450:400c:c06::bc
23 Discovered open port 5229/tcp on 2a00:1450:400c:c06::bc
24 Discovered open port 5239/tcp on 2a00:1450:400c:c06::bc
25 Discovered open port 5237/tcp on 2a00:1450:400c:c06::bc
26
```

7.2.1 *Princípios básicos do processo de MITM*

Qualquer que seja o *software* a utilizar para realizar as tarefas de intercepção é necessário configurar o IP da *proxy* no DM, para que o tráfego possa ser analisado. Tal é feito através da linha de comandos que a Listagem 30 documenta. Para reverter o processo apresentado em 30, basta executar o comando presente na listagem 31.

O esquema de rede utilizado é ilustrado na figura 14 (Secção 4.3.9 / página 82).

Listagem 30: Configurar o IP da proxy no dispositivo móvel

```
1
2       $ settings put global http_proxy <proxy_hostname>:<proxy_port>
```

Listagem 31: Repor o default gateway

```
1
2       $ settings put global http_proxy :0
```

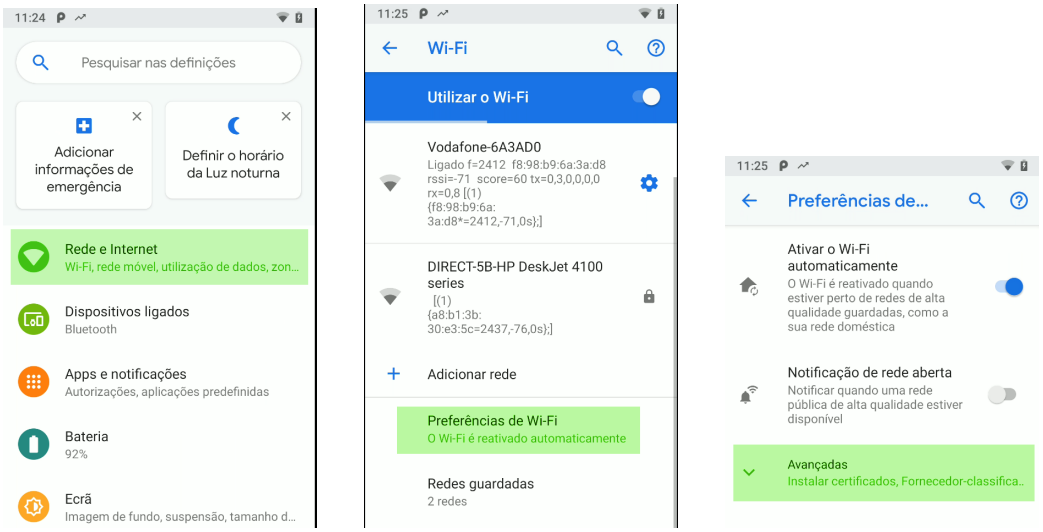
7.2.2 *Procedimento utilizado*

Começamos por intercetar de forma direta o tráfego, capturando todos os pedidos/respostas entre a aplicação e um destino remoto. Esta abordagem fracassou, revelando que a aplicação deteta que o tráfego está a ser intercetado, e sem explicitar a situação, notifica o utilizador com a mensagem "Impossível ligar à Internet". Verifique as configurações de rede". Tal indicia que a aplicação implementa medidas de segurança contra este tipo de ataque.

Por forma a se conseguir ultrapassar os mecanismos de proteção contra ataques de MITM, foi exportado o certificado do *BurpSuite* para o DM. Por forma a garantir que as configurações do laboratório foram corretamente efetuadas, em primeiro lugar foi testado através do browser do DM a possibilidade de intercetar o tráfego. E foi constatado que o mesmo era possível de ser interceptado, tendo assim a certeza que o certificado que foi instalado no cliente estava a ser utilizado e a *proxy BurpSuite* estava a intercetar tráfego.

Seguidamente, apresenta-se uma descrição, assente em imagens, do processo de instalação do certificado. Importa notar que o processo de instalação poderá depender do DM, da versão Android e do dispositivo empregue.

1. Começamos por consultar as definições do dispositivo, selecionamos a opção *Rede e Internet*, conforme demonstrado pela figura 28a.
2. Em seguida selecionamos as "Preferências de *Wi-Fi*", figura 28b.
3. A opção "Avançadas"figura 28c, permite-nos ter acesso à opção para instalar certificados.
4. Prosseguimos selecionando a opção "Instalar certificados", como se pode observar na figura 29a.
5. Procuramos no diretório local do DM no qual guardamos o certificado, selecionando o certificado anteriormente exportado, figura 29b.



(a) Passo 1 (b) Passo 2 (c) Passo 3

Figura 28: Instalar o certificado Burp, no DM (parte 1/3)

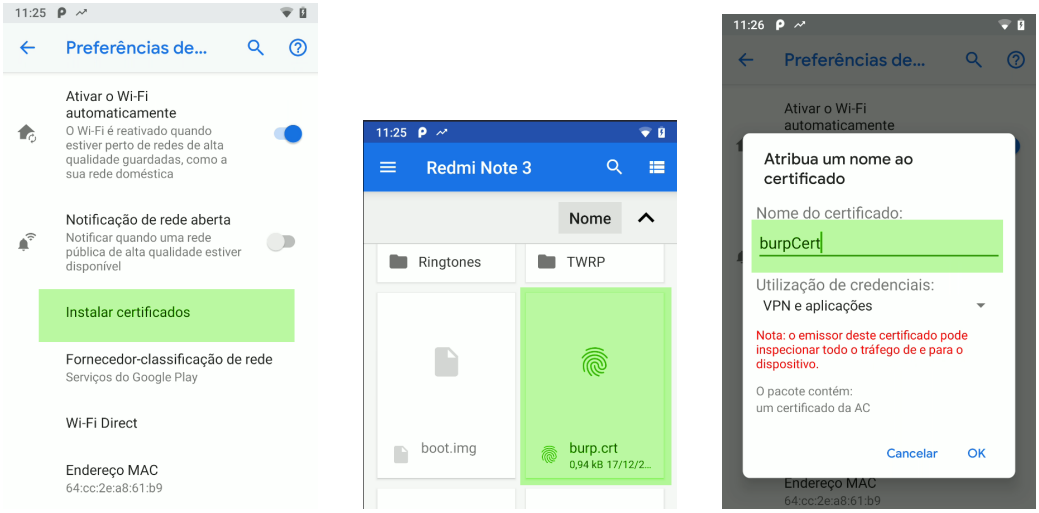
6. É necessário atribuir um nome ao certificado que será instalado, conforme figura 29c. O nome escolhido será guardado no repositório de certificados do utilizador e é através deste que podemos mais tarde consultar se foi corretamente instalado. De referir que o certificado que estamos a instalar não é considerado um certificado de sistema e como tal é guardado nos certificados de utilizador.

7. Por último o utilizador é informado se o certificado foi ou não corretamente instalado, figura 30a página 139.

Instalar o nosso próprio certificado não foi suficiente para conseguir ultrapassar o mecanismo de segurança existente contra ataques de MITM. A razão do mesmo prende-se com a forma como o certificado é instalado na *certificate store* do DM. De referir que desde o *Android* 7 é apenas possível instalar o certificado na "loja do sistema"*(certificate system store)*, com acesso *root* ao dispositivo.

Tendo acesso *root* ao dispositivo foi também instalado o certificado do *BurpSuite* com um nível de confiança de sistema. Para isso foi utilizado o procedimento referido pela PortSwigger[2] a adoptar em dispositivos com versão superior a *Android* 7.0 (Nougat). Sendo a versão do DM de teste *Android* 9.0 foi realizado o procedimento de forma a reconhecer o certificado do *BurpSuite* como legítimo pela CA *System*

2 Configurar BurpSuite após Android Nougat - Disponível em: https://blog.ropnop.com/configuring-burp-suite-with-android-nougat - Acedido a 31 Jan 2024



(a) Passo 4 (b) Passo 5 (c) Passo 6

Figura 29: Instalar o certificado Burp, no DM (parte 2/3)

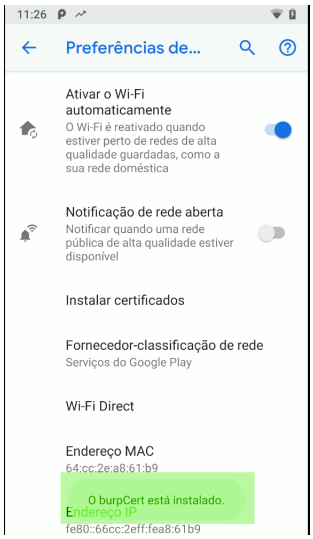
Store do *Android*. Podemos ver o resultado do procedimento através da figura 31, o qual confirma que o sistema reconhece o nosso certificado como legítimo.

Após instalar o certificado na *system store* foi possível intercetar as comunicações da aplicação Bolt, através da *proxy BurpSuite*.

No entanto esta situação cria um outro problema[3], que ocorre desde a versão 99 do *Google Chrome*, independetemente da ferramenta de MITM empregue. A limitação encontrada pode ser resumida da seguinte forma, ao colocar o nosso certificado (*PortSwigger*) na *system store* é possível intercetar tráfego da aplicação *Bolt*, mas deixa de ser possível intercetar tráfego no *Google Chrome* posterior à versão 99. Se for instalado o certificado apenas no *user store* é possível intercetar tráfego do *Chrome* mas não da aplicação, a solução utilizada para contornar este problema modificar as *flags* em uso pelo *Chrome*. Ficamos assim com total controlo na interceção das comunicações que são efetuadas tanto sobre a aplicação *Bolt*, como a utilização de serviços que necessitem de acesso através do browser (Chrome - `com.android.chrome`).

Outra alteração na instalação de certificados que foi introduzida no *Android* 11, estabelece que é necessário verificar quem requereu a instalação do certificado. Se não for iniciada pelas definições de sistema da aplicação, a instalação é recusada e alertado o utilizador que não pode instalar certificados de CA.

3 Chrome 99 e certificados self-signed - Disponível em: https://httptoolkit.com/blog/chrome-android-certificate-transparency/- Acedido a 1 Fev. 2024



(a) Passo 7

Figura 30: Instalar o certificado Burp, no DM (parte 3/3)

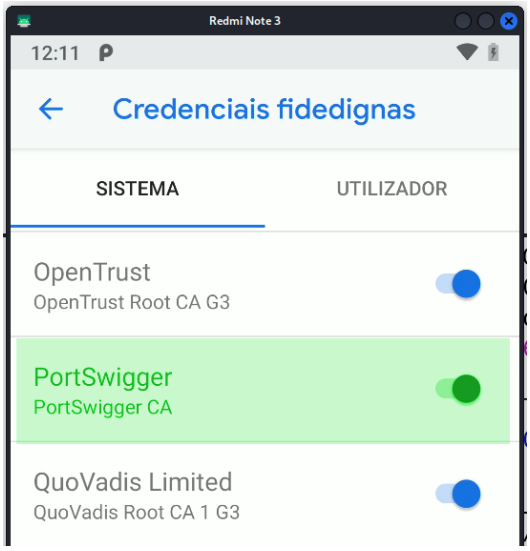


Figura 31: Certificado BurpSuite instalado na System Store

7.2.3 *Objection*

Existem outras formas de ultrapassar a proteção contra MITM existentes, isto para dispositivos que não exista a possibilidade de instalar um certificado na *System Store*. Existem diversos *scripts* genéricos em *javascript* que podem ser utilizados através do *Frida*[4]. Para facilitar este processo optou-se por recorrer ao *Objection*, dado que esta ferramenta disponibiliza um comando específico para executar *scripts* genéricos. O próprio *Objection* recorre ao agente *Frida*, ao injetar no processo associado à aplicação em análise, podemos observar o mesmo na listagem 32.

Listagem 32: Iniciar uma consola interativa para explorar a aplicação

```
1 $ objection --serial 424ac404 --gadget ee.mtakso.client explore
```

Como podemos ver na listagem 32, é necessário especificar o *package* e o dispositivo para interagir com a aplicação. O comando para desativar o processo de *sslpinning* está ilustrado na Listagem 33.

Listagem 33: Comando para desativar Certificate Pinning

```
1 ee.mtakso.client on (Xiaomi: 9) [usb] # android sslpinning disable
```

Podemos ver na figura 32 o resultado, após a desativação do *SSL Pinning* ao iniciar a aplicação. O *output* da ferramenta revela que está a ser reescrito o resultado das funções associadas ao processo de SSL/TLS, permitindo ter acesso ao conteúdo das mensagens que estão a ser trocadas.

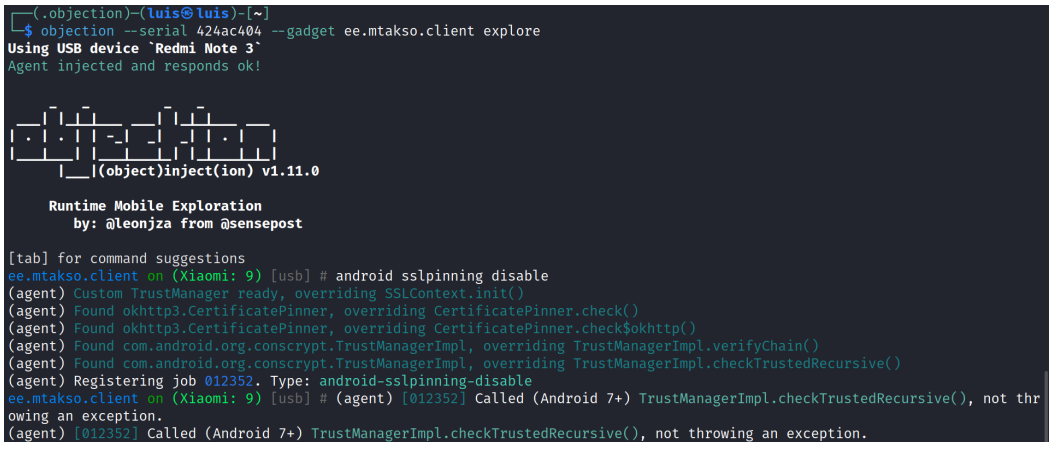


Figura 32: Desativar SSL Pinning

4 CodeShare, Frida - Disponível em: https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/ - Acedido em 3 Mar. 2024

Após o processo anterior podemos então configurar o *BurpSuite* de forma a intercetar o tráfego pelo dispositivo, uma vez que as medidas que restringiam o seu acesso foram suprimidas.

Dessa forma temos apenas que alterar o fluxo de tráfego gerado pelo dispositivo, para ser reencaminhado para a máquina do analista. O tráfego irá então ser analisado através do *BurpSuite*, presente na máquina do analista. Podemos observar na figura 14, uma representação do esquema de rede que foi utilizado para realizar os testes e na figura 15 as configurações do *BurpSuite*.

7.2.4 *Iniciar a aplicação pela primeira vez*

Ao iniciar a aplicação (*Bolt*) pela primeira vez após a instalação da aplicação, notamos que a aplicação tenta ter acesso ao seu último estado/uso.

A figura 33 (página 142), mostra os dados enviados na fase inicial pela aplicação ao servidor remoto.

Dados recolhidos:

- Versão da aplicação;
- Nome do dispositivo;
- Versão do OS;
- Canal de instalação da aplicação *(googleplay)*
- Tipo de dispositivo;
- Idioma;
- Fuso horário;

Em seguida é enviado para o serviço *Firebase Remote Config*, informação que descreve a aplicação instalada no dispositivo. A esta informação é devolvida uma resposta que define o comportamento da aplicação, pretendida pelos produtores da aplicação. A resposta consiste num conjunto de parâmetros, expressos em formato JSON, que definem o comportamento da aplicação (figura 34), nomeadamente ao nível das métricas de desempenho que deve ser recolhidas pela aplicação e enviadas para a *firebase*.

Existe no entanto informação cifrada, em comunicações posteriores para a *Firebase*, não tendo sido possível ter acesso ao seu respetivo conteúdo.

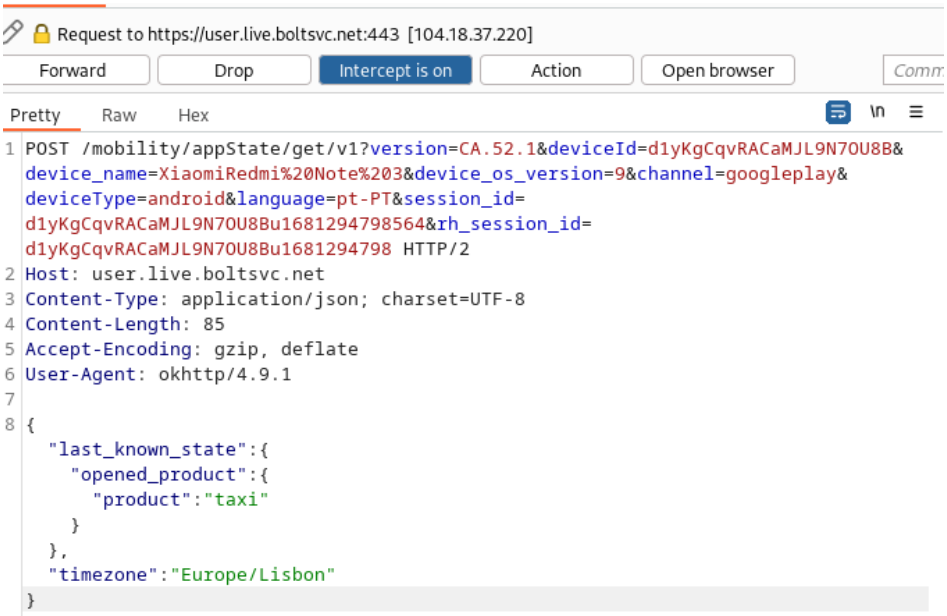


Figura 33: Primeira iteração com a aplicação

7.2.5 *Registar número de telefone*

Ao inserir o número de telefone é também enviado para o *Host: node.bolt.eu*, informação referente ao DM, bem como as coordenadas GPS atualizadas, tal como podemos ver na figura 35.

De referir que a permissão é requerida ao seu utilizador para aceder à localização do DM ao iniciar a aplicação pela primeira vez. Sem esta permissão ser aceite pelo utilizador não é possível o acesso à localização.

Durante o processo de registo é necessário validar que temos acesso ao número que digitamos como sendo nosso. Como tal foi testado o mesmo para verificar de que forma o processo é efetuado. Nos testes efetuados não foi possível obter o código de acesso, sem ter acesso ao respetivo número de telefone que foi utilizado para o registo. Podemos ver que na resposta recebida (figura 36), não é revelado o código a ser utilizado para confirmar o registo do número de telefone, apenas o método utilizado para enviar o mesmo ("SMS").

Foram realizadas tentativas de envio de múltiplos pedidos de códigos de confirmação. Esta ação é desempenhada através do *Repeater* disponível no *BurpSuite*, o qual nos permite modificar o *"request"* a ser enviado para o servidor e observar a resposta do mesmo. Estas tentativas tinham o objetivo de verificar se era possível algum tipo de negação associado a este serviço. O resultado das tentativas revelou-se

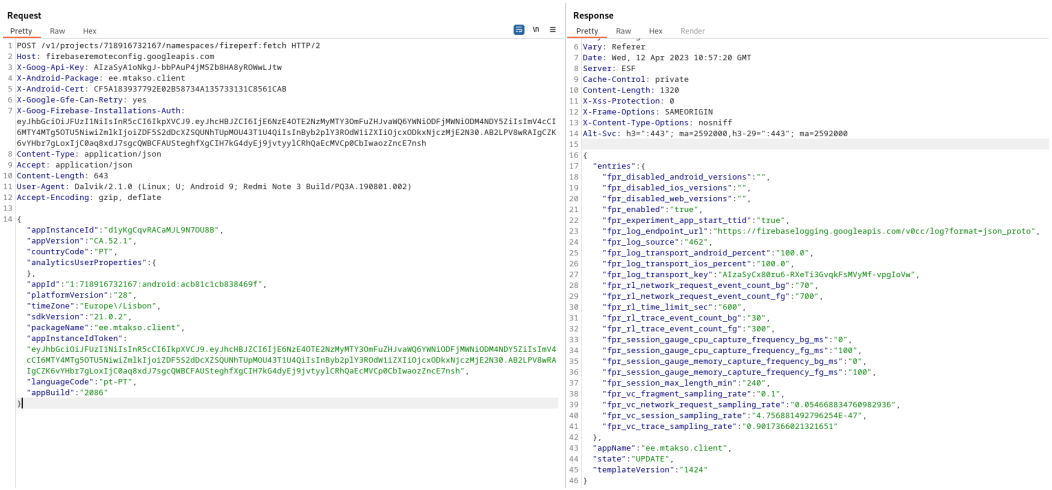


Figura 34: Definição de configuração e opções de desempenho registadas pela Firebase

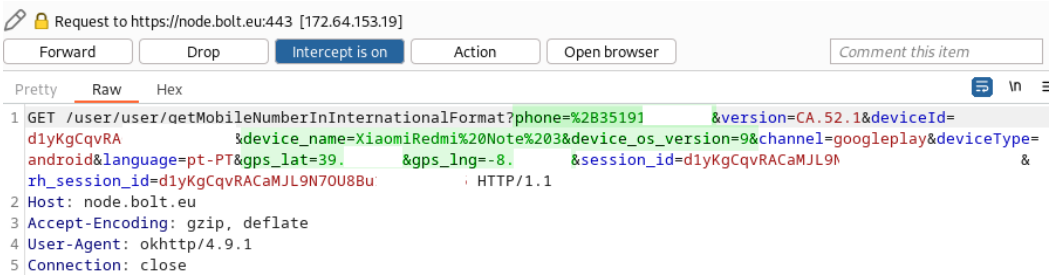


Figura 35: Registar número de telefone

infrutífero, tendo originado a notificação que foi efetuado um número excessivo de pedidos (figura 37).

Através da figura 36, observar-se que o intervalo de reenvio do código é de 20 segundos, no entanto quisemos verificar se o intervalo era ou não respeitado. O resultado do mesmo revelou este é respeitado e como tal não é possível realizar um ataque de negação de serviço através do serviço de "SMS".

Este sistema tem também falhas uma vez que o mesmo utilizador pode de forma incorreta, através do *BurpSuite* ou outra *proxy* de intercessão, efetuar diversos pedidos com diferentes números de telefone para envio de pedidos de activação. De salientar que basta para isso utilizar um número distinto em cada vez que efetua um novo pedido. No entanto este tipo de funcionalidade por si só não representa uma quebra de segurança, tanto ao utilizador ou sistema. Se eventualmente múltiplos utilizadores receberem pedidos de activação por parte da *Bolt* pode ser considerado em parte uma prática de envio de mensagens de *spam*. O que pode em certa parte trazer um má conotação ao nome da aplicação/empresa.

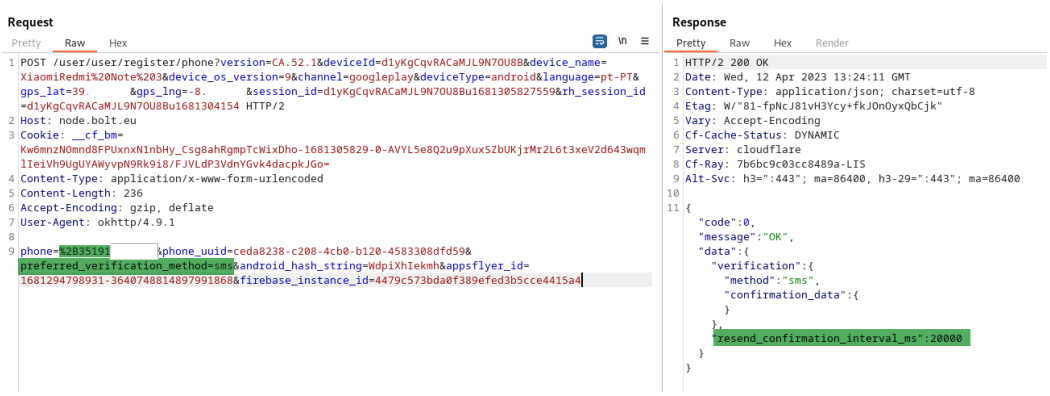


Figura 36: Obter código de registo para número de telefone

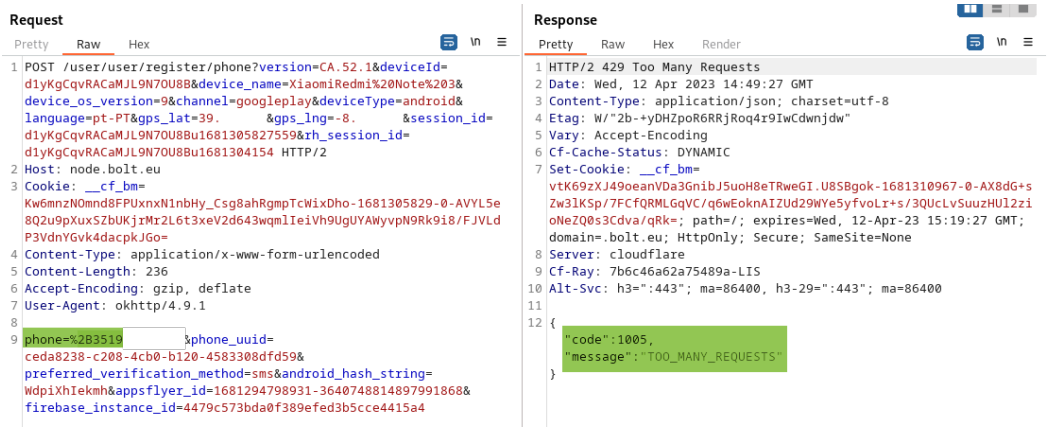


Figura 37: Envio de múltiplos códigos de ativação

Podemos ver um exemplo da situação descrita anteriormente na figura 38 com o número de telefone ("*+351 91 <omisso>*"), o qual foi enviado com sucesso a mensagem.
Foi testado com recurso a um segundo número de telefone, o qual confirma que recebemos a mensagem com o código de activação para o mesmo. No entanto a mensagem acompanha com um ID que identifica quem requereu o código de ativação. Para a mesma mensagem foi também testado sem sucesso, efetuar alteração do conteúdo da mensagem enviada para o cliente. O ID referido está presente no campo `"android_hash_string"`, e este é sempre enviado em conjunto para os diversos números utilizados.

Após testes sucessivos de acordo com o intervalo de tempo de 20 segundos, surgiu a mensagem que nos impossibilita de continuar a explorar esta vertente. Podemos verificar na figura 39 ambas as mensagens que foram obtidas após os testes anteriores. Para podermos obter de novo um código é necessário remover e instalar de novo a aplicação, após esse processo podemos voltar a obter novos códigos de activação. O que reduz a relevância deste achado, tornando o processo de exploração

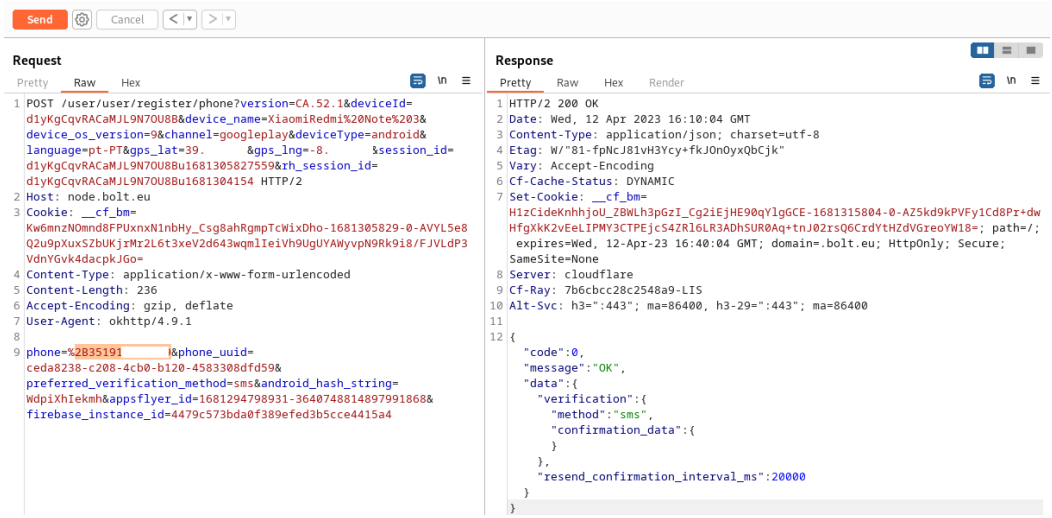


Figura 38: Envio de spam para número aleatório

mais trabalhoso o qual se traduz num menor risco de o mesmo ser explorado por potenciais atacantes.

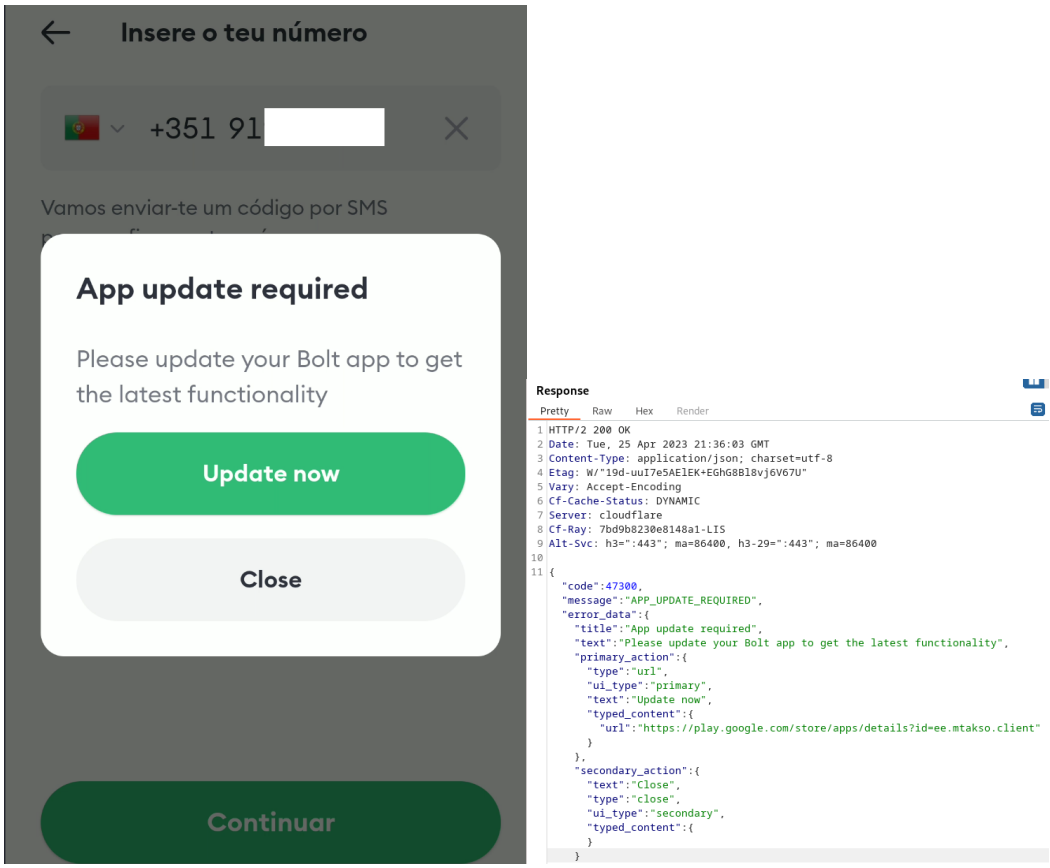
7.2.6 *Interação com o mapa*

A aplicação ao interagir com o mapa envia, conforme estabelecido periodicamente informação relativa à posição do seu utilizador para o servidor "*user.live.boltsvc.net*".

Podemos ver o que está a ser enviado para este servidor, e perceber qual a sua função para o funcionamento da aplicação. O servidor para o qual é enviada esta informação (*user.live.boltsvc.net*) - é utilizado para manter em tempo real a localização de todos os clientes que têm a aplicação ativa. Desta forma podemos afirmar que a *Bolt* tem acesso em tempo real a toda a informação de localização dos seus utilizadores. Os servidores estão alojados numa Content Delivery Network (CDN) da *cloudfare*, como tal é difícil obter mais informação sobre os servidores, além dos IP's utilizados.

Contudo, para além da informação de geolocalização, a aplicação envia um volume muito significativo de dados para o servidor.

Devido à grande extensão de informação que é enviada, optámos por dividir os dados recolhidos em categorias. Foram também removidos dados duplicados e irrelevantes para esta análise, no entanto uma cópia pode ser encontrada nos documentos no apêndice 53. De salientar também que foram efetuadas diversas



(a) Interface Bolt (b) Resposta BurpSuite

Figura 39: Interface App e resposta após envio de multiplos códigos de ativação

recolhas de informação em diferentes instâncias o que pode levar a dados dispares durante as diferentes análises efetuadas.

Dados recolhidos enquadram-se nas seguintes categorias:

- Rede;
- Informação sobre o dispositivo:
  - *Firmware, Software*;
  - *Sensores e Bateria*;
  - *Paths*;
- *Timezone*, Idioma e Localização;

O elevado detalhe de dados recolhidos pela aplicação exigiu um resumo e enquadramento pela ordem definida anteriormente.

Os resultados apresentados na listagem 34, respeitante aos dados de rede, respeitam a formatação que foi obtida na análise Burp.

Listagem 34: Dados recolhidos - Rede

```
"json_data": {
"---Removido para brevidade---":...,
    "49": "e2:b9:e5:b5:c9:5b",
    "50": "\"MEO-<omitido>\"",
    "---Removido para brevidade---":...,
    "55": "192.168.1.72",
    "---Removido para brevidade---":...,
    "59": "192.168.1.254",
    "61": 3600,
    "62": "WIFI",
    "65": "64:CC:2E:A8:61:B9",
    "66": "fe80::66cc:2eff:fea8:61b9",
"---Removido para brevidade---":...,
}
```

Os dados recolhidos são enviados sobre o formato JSON. Seguidamente identifica-se o significado de cada campo.

Legenda da listagem 34:

- **"49, 65"**: Endereço MAC Router e endereço MAC do dispositivo móvel;
- **"50"**: SSID Router;
- **"59"**: Endereço IPv4 do Gateway para Internet;
- **"62"**: Modo de ligação à rede local;
- **"55, 66"**: Endereço de rede IPv4 e IPv6, dispositivo móvel;

Na listagem anterior podemos observar que são obtidos dados de rede em excesso para as funções necessárias ao funcionamento da aplicação. Informação como *SSID*, que pode servir para possível geolocalização do dispositivo de rede, endereços de MAC e IP locais, tanto do DM como *router* são irrelevantes tanto para o funcionamento da aplicação como para tarefas de *debug* de eventuais erros que possam ser usados pela equipa de desenvolvimento da aplicação.

É também recolhida a lista de sensores que são utilizados pelo DM, para posicionamento da localização do cliente. Podemos observar um excerto do detalhe de informação respeitante à mesma na seguinte listagem 36.

Mais uma vez o nível de detalhe da informação relativa aos sensores existentes, é no entender do autor excessivo. Esta informação não será útil para a *Bolt* realizar os serviços para os quais a aplicação é utilizada, nem aparentar ser necessária para relatórios de erros.

Quanto à informação de bateria, poderá ser útil o seu nível, para efeitos de notificar o seu utilizador do baixo nível da mesma, ou para reduzir recursos de

Listagem 35: Dados recolhidos - Dispositivo - Firmware, Software, CPU

```
"json_data": {
        "1": "Xiaomi",
        "2": "Redmi Note 3",
        "7": "kenzo",
        "8":
        ↪  "Xiaomi/kenzo/kenzo:6.0.1/MMB29M/V8.2.1.0.MHOCNDL:user/release-keys",
        "9": "qcom",
        "10": "PQ3A.190801.002",
        "11":
        ↪  "976_GEN_PACK-1.103654.1.105572.1,976_GEN_PACK-1.103654.1.105572.1",
        "13": 28,
        "14": "9",
        "17": 20654321664,
        "20": 26283376640,
"---Removido para brevidade---":...,
        "22": 480,
        "23": 1920,
        "24": 3.0,
        "25": 1080,
        "26": 391.885,
        "27": 381.0,
        "47": "aarch64",
...,"---Removido para brevidade---":...,
        }
```

utilização. Ao contrário de outras aplicações móveis no mesmo segmento de mercado (Uber) que são acusadas de subirem as tarifas quando detetam que o cliente está com pouca bateria no DM[5], não existem trabalhos científicos que sustentem a mesma prática por parte da *Bolt*. Toda a restante informação aparenta novamente ser excessiva para a funcionalidade da aplicação.

Na lista 37 podemos observar que a aplicação recolhe informação sobre todos os diretórios aos quais pode ter acesso, bem como todos os diretórios necessários para a execução das funções da mesma. Esta informação pode ser útil à aplicação para gerir e otimizar o seu desempenho. No entanto não existe a necessidade de enviar para um ponto de acesso remoto esta informação, a menos que exista um evento de erro/notificação decorrente do uso da aplicação.

De seguida agrupamos os campos que foram considerados como contendo dados excessivos nas listagens anteriormente descritas.

Legenda da listagem 35, 36, 37 :

- **"1-14, 47"**: Diversas especificações do dispositivo (*Firmware* OS), arquitetura do CPU.
- **"17-20"**: Espaço em disco disponível e espaço máximo;

5 Uber, custo associado ao nível de bateria https://www.forbes.com/sites/amitchowdhry/2016/05/25/uber-low-battery/ e https://www.vice.com/en/article/m7beq8/uber-surge-pricing-phone-battery - acedido a 24 Jan 2024

Listagem 36: Dados recolhidos - Sensores e Bateria

```
"json_data": {
...,"---Removido para brevidade---":...,
        "72": {
                "BMI160 Accelerometer": "BOSCH/2061000",
                "YAS537 Magnetometer": "Yamaha/35193090",
                "BMI160 Gyroscope": "BOSCH/2061000",
                  "Linear Acceleration": "QTI/2",
                "Rotation Vector": "QTI/2",
                "Step Detector": "QTI/2",
                "Step Counter": "QTI/2",
                "GeoMagnetic Rotation Vector": "QTI/2",
                "Orientation": "QTI/2",
                "Tilt Detector": "QTI/2",
                "Motion Accel": "QTI/2",
        },
        "73": {
                "level": 55,
                "scale": 100,
                "status": 2,
                "charging": true,
                "chargeplug": 2,
                "temperature": 327,
                "voltage": 3737,
                "health": 2
        },
"---Removido para brevidade---":...,
}
```

- **"22-23, 24"**: Dimensões de ecrã e resolução, RAM;
- **"69"**: Diretórios e arquivos SO Android;
- **"72"**: Hardware, sensores de movimento e localização;
- **"73"**: Bateria;

A informação que é apresentada na listagem 38, é útil para as funções da aplicação, desta fazem parte o fuso horário, idioma e a localização do DM. Como tal é considerada como pertinente para desempenhar as funções requeridas pela aplicação.

7.2.7 *Pedido de transporte*

Ao procurar por um destino é apresentado um conjunto de taxis/veículos, disponíveis para realizar a viagem. Na interface Bolt é apenas apresentado o preço para cada categoria de veículos automóveis disponíveis *(Bolt, Economy, XL, Pet, Electric)*, de acordo com o veículo mais perto.

Listagem 37: Dados recolhidos - Paths

```
"json_data": {
...,"---Removido para brevidade---":...,
"69": [
        "/system/framework/services.jar:/system/framework/ethernet-service.jar:/
        ↪  system/framework/wifi-service.jar:/system/framework/com.android.loca
        ↪  tion.provider.jar",
        "/sbin:/system/sbin:/system/bin:/system/xbin:/odm/bin:/vendor/bin:/vendo
        ↪  r/xbin", "/cache", "/data", "/system/etc/terminfo", "/mnt/asec",
        ↪  "/sdcard", "/system/app","/storage", "/system",
        ↪  "/data/cache",        "/system/framework/core-oj.jar:/system/framewo
        ↪  rk/core-libart.jar:/system/framework/conscrypt.jar:/system/framework
        ↪  /okhttp.jar:/system/framework/bouncycastle.jar:
 ...,"---Removido para brevidade---":..., ],
}
```

Listagem 38: Dados recolhidos - Fuso horário, Idioma e localização

```
        "json_data": {
        ...,"---Removido para brevidade---":...,
                "44": "pt",
                "45": "Europe/Lisbon",
        ...,"---Removido para brevidade---":...,
                "85": 39.602726,
                "86": -8.835962
        }
```

Através da API, podemos ver todos os veículos disponíveis para realizar a viagem, de cada uma das categorias referidas anteriormente e a sua posição (GPS), conforme ilustrado figura 40.

Não foi efetuado o cruzamento de informação no entanto, é possível saber qual o condutor através do campo `id`, que identifica o respetivo condutor. Além dessa informação temos acesso à sua posição, através das coordenadas GPS, bem como o campo `bearing`, que representa a direção do movimento do veículo. A informação aqui descrita está presente na figura 40. Esta funcionalidade permite a um interveniente registar o deslocamento de um táxi/veículo e/ou escolher consoante o `"id"` o condutor da viatura. Não que isso provoque diretamente algum tipo de incongruência com o que está escrito na política de privacidade, no entanto, um possível atacante fica assim informado que o condutor se encontra em serviço. No possível caso de conhecer a sua morada, pode eventualmente facilitar um assalto da sua casa ou em caso de roubo da viatura/condutor.

O fator de pagamento de uma viagem é determinante para a qualidade da aplicação em estudo, uma vez que é este que determina a viabilidade do seu sucesso. O preço por viagem é um dos fatores que determina a satisfação e a reutilização do serviço

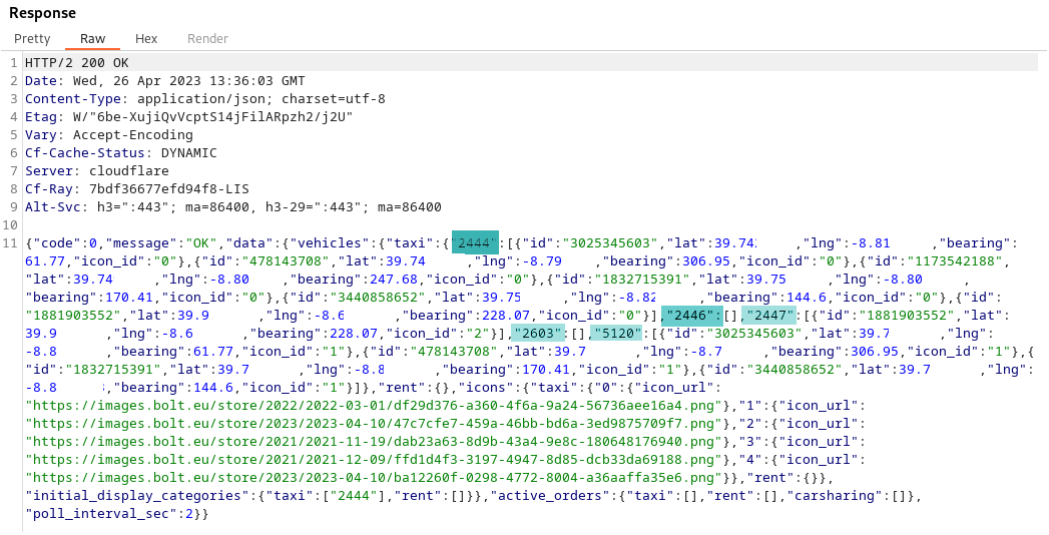


Figura 40: Lista de Veículos Disponíveis

pelo cliente. No entanto, se não houver lucro para distribuir entre os parceiros, o negócio não será viável.

Como tal, é fundamental testar o mesmo na eventualidade de existir a possibilidade de alterar o valor da viagem de forma indevida. Para isso, mais uma vez, recorrendo ao BurpSuite, obtivemos o pedido que regista o momento em que é enviado o pedido para o servidor com a informação relativa à rota pretendida, em conjunto com o preço da viagem considerado.

Antes mesmo de reenviar este pedido, podemos desde logo verificar o campo `price_lock_hash`. Este campo é utilizado para garantir a integridade do valor acordado entre cliente e prestador de serviço, sem que este seja alterado de forma inadvertida ou intencional.

Na figura 41, está representado o teste que foi efetuado, com a alteração do campo `price_accepted` de 37.28€ para 0.28€. Podemos observar o resultado do lado direito da figura 41, que representa a resposta (*Response*) ao pedido (*Request*), seguindo a interface do *BurpSuite*.

A função de *hash* é calculada de novo com o novo valor alterado, e na resposta a comparação é efetuada com a *hash* original, presente no campo `price_lock_hash`. Após efetuar a comparação recebemos um código de erro "7001", com a mensagem `Price_lock_invalid`. Demonstrando assim que a *hash* garante a integridade do valor a pagar, não permitindo alteração do valor.

O que demonstra que a operação de *hash* é realizada antes de apresentar ao cliente o valor a pagar, permitindo apenas aceitar o valor proposto pela plataforma.

De seguida através da análise estática é necessário localizar se a função responsável pelo cálculo de *hash* é feita do lado do cliente. No caso de não ser possível localizar a mesma no código da aplicação, esta será feita do lado do servidor.

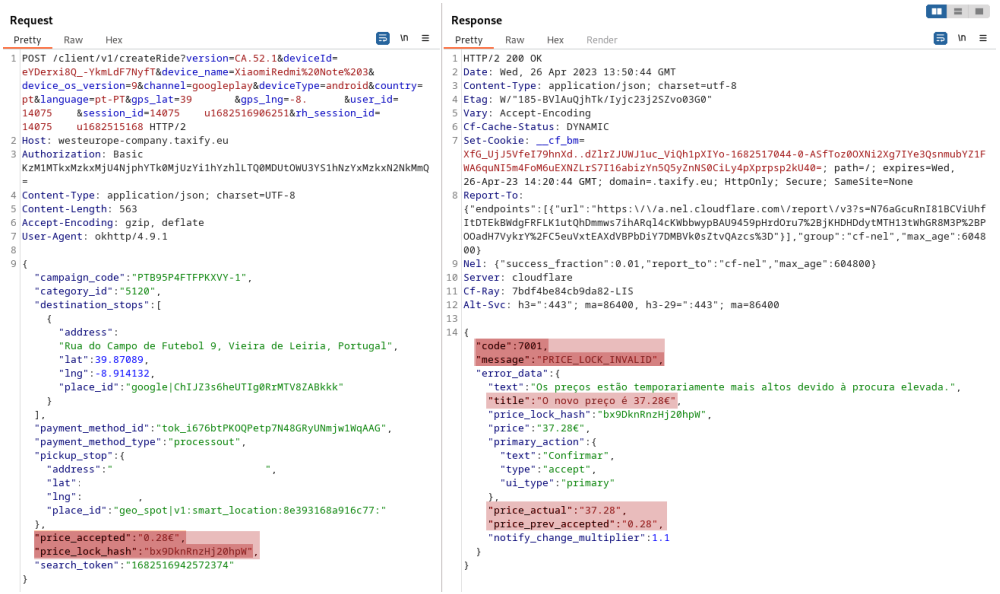


Figura 41: Alteração de preço da viagem

7.2.8 *Dados sensíveis partilhados com entidades terceiras*

Todos os pedidos (*requests*) que são efetuados pela aplicação *Bolt* que não são enviados diretamente ao servidor hospedado pela aplicação devem ser verificados quanto à partilha de comunicação sensível (Personally Identifiable Information (PII)), tais como o caso de *trackers* ou serviços de publicidade.

Por forma a estudar o conteúdo partilhado, é necessário identificar os *trackers* existentes na aplicação *Bolt*, para isso foi criada a Tabela 26.

TRACKER: MIXPANEL O *tracker Mixpanel*, com o `host: "api.mixpanel.com"`, não existe informação considerada como PII, pois não existe relacionamento direto entre uma pessoa, a uma morada. A informação que é trocada na comunicação pode no entanto ser considerada como PII pela forma como indiretamente é utilizada, além do que mais uma vez, a informação que pode ser considerada como excessiva.

Para este *Host*, a informação enviada pela aplicação é codificada em Base64 antes de ser enviada, o que melhora a sua confidencialidade mas não a garante.

Tabela 26: *Tracker's* da aplicação Bolt

Tracker	**Categoria do tracker**
AppsFlyer	Analítica
CleverTap	Analítica, criação de perfil e localização
Facebook Analytics, Flipper, Login, Notifications, Share	Analítica e identificação
Google CrashLytics	Relatórios de erros
Google Firebase Analytics	Analítica
BugSnag	Relatórios de erros
MixPanel	Analítica, Publicidade

Falamos do envio de informação de localização, enviada em conjunto com o identificador do utilizador *Bolt* (*'user_lat*:39.'[valor omisso], *'user_lng*:-8.'[valor omisso],..., *'bolt_user_id'*: [valor omisso]). Esta informação pode ser utilizada para identificar a movimentação e localização atual do cliente *Bolt* e assim indiretamente com base na localização mais frequente perceber hábitos e identificar a sua morada. Isto porque em conjunto com o campo *'mp_session_start_sec:1706700576'*, que representa o tempo em *UNIX Timestamp* (em segundos) a cada nova atualização despoletada pelo *tracker* à aplicação é possível identificar o posicionamento do utilizador ao longo do dia.

Os restantes dados enviados consistem principalmente em dados analíticos relacionados ao uso da aplicação *Bolt*. Contém informações sobre o dispositivo (como fabricante, modelo, versão do sistema operativo), algumas configurações da aplicação (idioma, configurações do mapa, etc.) e dados relacionados a eventos específicos da aplicação. No entanto este tipo de informação recolhida é o esperado para este *tracker* em específico.

TRACKER: CLEVERTAP No caso do *tracker CleverTap*, os dados são enviados para o *host: eu1.clevertap-prod.com* em texto simples, sem qualquer tipo de codificação ou cifragem, no entanto não existe informação sensível ou pessoal que possa identificar o utilizador.

A informação existente identifica com detalhe característica do DM, desde o OS ao espaço em disco disponível. De referir que existem campos que fazem parte da estrutura *Json* do envio de informações a este *Host* que podiam facilmente ser utilizados para identificar o utilizador. Falámos dos campos `"user_last_name"`, `user_first_name`, `user_email` e `user_phone`, no entanto o seu conteúdo encontra-se vazio. Foram analisadas comunicações posteriores, no entanto não houve alteração

no conteúdo da informação enviada, no que diz respeito aos campos com conteúdo sensível referidos anteriormente.

TRACKER: APPSFLYER Para o *tracker AppsFlyer*, os dados são enviados para o *host: 'launches.appsflyer.com'*. Os dados enviados para o *host* anteriormente referidos estão cifrados, não permitindo a sua leitura e análise. Está assim garantida a sua confidencialidade, no entanto não é possível identificar se existe ou não troca de informação PII através desta técnica de análise.

TRACKER: FACEBOOK A *Bolt* permite a ligação com uma conta pré-existente ao *Facebook*, que por sua vez detém um conjunto de *trackers* próprios com funções distintas que agregam informação para ser usada para analítica. O que resulta no envio de informação para diferentes *host's*. Para simplificar a explanação da informação recolhida (PII) recolhida foi criada a Tabela 27.

Tabela 27: Dados recolhidos pelo *tracker Facebook*

Host	**Cifragem**	**Dados sensíveis recolhidos**
`graph.facebook.com`	Dados cifrados	Não foi possível a sua leitura.
`m.facebook.com`	Codificados (*base64*) e simples	Não existem dados sensíveis. Dados de *login* com a *webApp facebook*, através de *cookies* com o *Facebook ID, app ID, user ID.*
`lm.facebook.com`	Codificados (base64) e cifrado (HMAC-SHA256)	Não existe informação PII, a única referência à app da Bolt é o nome do *package*(ee.mtakso.client).
`gateway.facebook.com`	Codificados (*base64*)	Não existe informação sensível (PII).

Podemos afirmar que não existe informação sensível/pessoal que é trocada entre o *Facebook* e a *Bolt*. O intuito da comunicação é fornecer informação que consiga autenticar o utilizador através do seu endereço de correio eletrónico ou número de telefone. Não existe envio de informação ligada a coordenadas/localização nos teste que foram efetuados.

TRACKER: GOOGLE FIREBASE ANALYTICS O *host* responsável por tratar a informação referente a este *tracker* é o `firebaseremoteconfig.googleapis.com`. A informação recolhida recai sobre as definições de software do DM e gestão de espaço de disco (do DM), existência de ligação ao *Facebook* assim como a versão da

aplicação *Bolt*. A única informação que identifica o seu utilizador é o país de onde foi realizada a instalação, e a língua utilizada na aplicação *Bolt*.

TRACKER: GOOGLE CRASHLYTICS O *tracker Google CrashLytics* é um ferramenta de relatórios de falha em tempo real, mantida pela *Firebase*, que pretende priorizar e corrigir as falhas mais comuns de aplicações móveis. Utiliza o *host* `firebase-settings.crashlytics.com`, o seu uso está ligado à obtenção das definições a serem usadas pela firebase na criação de relatórios. Ou seja esta define a periodicidade na criação dos relatórios, assim como as métricas que deveram ser utilizadas aquando da criação dos relatórios de erros.

TRACKER: BUGSNAG O *tracker Bugsnag* é utilizado para monitorização e registo de erros da aplicação, e posterior criação de relatórios baseados nos mesmos. O *host* observado para envio de informação foi o `sessions.bugsnag.com`. A informação recolhida revela apenas informação do software do dispositivo, o identificador utilizado regista a sessão do *Bugsnag*. Não foi encontrada qualquer informação com conteúdo sensível/privado que possa identificar o seu utilizador.

7.3 MOBILE DEEP LINKS

O funcionamento dos *deep links* dita a forma como são realizadas as comunicações entre aplicações *Android* 3.9.2. Uma aplicação pode comunicar com os diferentes componentes, através de um *Intent*. *Intent* é um objeto utilizado para troca de mensagens e é caracterizado por uma *"action"*, *"category"* e *"data"*. Os *deep links* despoletam um tipo específico de *intent* para permitir a comunicação entre aplicações web e aplicações móveis.

O uso de *deep links* assenta em duas fases distintas:

1. **Registo:** uma aplicação deve registar todos os seus URI's no SO durante a instalação. Os URI's são declarados no campo '`data`' para cada um dos *intent-filter* (`AndroidManifest.xml`).
2. **Endereçamento:** quando o *link* é selecionado , o SO móvel procura por correspondências em todos os *intent-filter*. Se o *link* coincidir com o URI da aplicação, o SO móvel inicia a aplicação correspondente.

Android Deep Links são uma forma de direcionar utilizadores para um conteúdo específico da aplicação, esteja esta instalada no DM ou não. No caso da aplicação alvo

não estar instalada, o utilizador é redirecionado para a página da aplicação na *Play Store*, sendo solicitado instalar ou descarregar a aplicação requerida. A funcionalidade de *Deep Links* é conseguida através da utilização de um URL específico que está diretamente ligado a conteúdo dentro da aplicação móvel. Estes existem para que a experiência do seu utilizador seja mais fluida, possibilitando o acesso direto ao conteúdo pretendido pelo utilizador, ao invés de ser redirecionado para a página inicial da aplicação. O que torna a navegação mais rápida, fluida e eficiente para o seu utilizador.

Android suporta dois tipos de *deep links*:

- **Custom URL schemes**: são *deep links* que utilizam esquemas específicos da aplicação, que não são verificados pelo SO. `(ex: bolt://)`.
- **Android App Links** `(Android >= 6.0(API 23))`: estes são verificados de forma automática pelo SO *Android* e contém o atributo `autoVerify` definido `(ex:http:// e https://)`.

Do ponto de vista da segurança é necessário garantir que o utilizador não seja redirecionado para uma página de *phishing* ou para uma aplicação que contenha *malware/spyware*. Este tipo de ação é conhecida como colisão de *deep link*, e pode ocorrer se outra aplicação instalada no DM estiver declarada para resolver o mesmo `intent`. O que permite a qualquer aplicação declarar controlo sobre o *deep link* destinada a outra aplicação. Em versões mais recentes *Android* no caso de dúvida sobre a aplicação alvo a tratar do *deep link* é requerido ao utilizador para escolher. No entanto também esta opção solução não é perfeita, pois o utilizador pode escolher uma aplicação maliciosa ao invés da legítima.

Para isso é preciso testar os *deep links* existentes (incluindo *App Links*), para diminuir a superficie de ataque da aplicação. Os riscos associados incluem *link hijacking*, exposição de funcionalidade critica ou sensível, entre outros.

De acordo com a documentação Android[6], existem dois tipos de *Intents*:

1. *Implicit Intent* - Estes são os *Deep Links* que redirecionam o utilizador para uma localização estática, em qualquer parte da aplicação sem especificar o componente exato a ser chamado. Estes são mais utilizados em *widgets* ou notificações, entre outros.

   Exemplo de *implicit intent com deep link* em Java:

6 Tipos Intents, Android Doc - Disponível em: https://developer.android.com/guide/components/intents-filters?hl=pt-br#Types- Acedido a 4 Ago. 2023

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("bolt://open/produto?id=0123"));
startActivity(intent);
```

O exemplo anterior é meramente ilustrativo e não representa o encontrado na aplicação em estudo. No código anterior o utilizador ao clicar no *link* do URI a aplicação irá receber o *intent* e utilizar o parâmetro `"id"` para determinar o produto a ser apresentado ao utilizador. Não é especificada a atividade exata, como tal será a lógica da aplicação, que determina a mesma tendo em conta parâmetros como URI e preferências do DM, entre outros.

2. *Explicit Intent* - Por outro lado, os *explicit deep links* redirecionam o utilizador para uma activity em específico para qualquer aplicação. Podem ser usados para iniciar um componente específico, seja este uma *activity, service* ou *broadcast receiver*.

   Exemplo de *explicit deep link* em Java:

```
Intent intent = new Intent(this, mainActivity.class);
intent.setData(Uri.parse("bolt://produto?id=0123"));
startActivity(intent);
```

   No excerto ilustrativo anterior, iria ser iniciada a aplicação *Bolt*, se esta estivesse instalada, e iria navegar para a página do produto identificado.

Algumas das vulnerabilidades criadas pelo uso de *Deep Links*, nomeadamente para *web intents* foram ultrapassadas pela implementação de medidas no *Android* 12[7]. Destas fazem parte a necessidade de o domínio destino do *intent* corresponder(*resolves*) a uma *activity* na nossa aplicação, para isso tem que ser previamente aprovada o acesso pelo domínio específico. O que impede a qualquer aplicação maliciosa de ter acesso a informação sensível que era passível de ser obtida, conhecendo apenas o domínio específico. Se tal acontecer os *web links* são abertos pelo *browser* por omissão e não por outras aplicações presentes no DM.

Identificação de Deep Links

Através da leitura do *Manifest* é possível retirar informação sobre os *Deep Links* existentes. Para nos auxiliar nesta tarefa foram utilizadas as ferramentas `get schemas` (Secção 4.2.12) e `App Link Verification` (Secção 4.2.4).

De seguida é apresentado um resumo dos *Deep Links* existentes na aplicação.

7 https://developer.android.com/about/versions/12/behavior-changes-all#web-intent-resolution - acedido a 20 de Julho de 2023

Deep Links declarados no Manifest:

- Activity: `ee.mtakso.client.newbase.deeplink.DeeplinkActivity`
  - `bolt://action`
  - `boltprelive://action`
  - `taxify://action`
  - `taxify://action/qr`
  - `https://scooters.taxify.eu`
  - `https://scooters.taxify.eu/qr`
  - `https://maps.google.com`
  - `geo://`

  Outras possíveis combinações, para a mesma activity[8] (apesar de estarem apenas declarados os anteriores, as seguintes combinações também são possíveis)
  - `bolt://`
  - `bolt://action/qr`
  - `bolt://scooters.taxify.eu`
  - `bolt://scooters.taxify.eu/qr`
  - `boltprelive://action/qr`
  - `boltprelive://scooters.taxify.eu/qr`
  - `https://action`
  - `https://action/qr`
  - `taxify://`
  - `taxify://action`
  - `taxify://scooters.taxify.eu`
  - `taxify://scooters.taxify.eu/qr`
- Activity: `ee.mtakso.client.newbase.deeplink.AppsFlyerDeeplinkActivity`
  - `https://bolt.onelink.me`

8 Android Docs, Guia deep links - Disponível em: https://developer.android.com/training/app-links/deep-linking?hl=pt-br#testing-filters - Acedido a 1 de Agosto de 2023

- Activity: `com.adyen.threeds2.internal.ui.activity.ChallengeActivity`
  - `adyen3ds2://ee.mtakso.client`
- Activity: `com.facebook.CustomTabActivity`
  - `fbconnect://cct.ee.mtakso.client`

Em seguida procedemos ao teste de todos os *deeplinks* identificados anteriormente. Para o efeito recorre-se ao utilitário 'am'[9] gestor de atividades do ADB – confirmando-se que os *deeplinks* são resolvidos de forma correta.

Listagem 39: Testes exemplos: Sintaxe para teste deep links de forma manual

```
1
2  # Invocar deeplinks de forma manual
3  $ adb shell am start -W -a "android.intent.action.VIEW" -d
   ↪  "https://bolt.onelink.me/" ee.mtakso.client
4
5  #Obter ficheiro DAL associado ao deep link: https://maps.google.com
6  $ adb shell am start -W -a "android.intent.action.VIEW" -d
   ↪  "https://maps.google.com/.well-known/assetlinks.json"
7
```

- URI's: `bolt://action`, `taxify://action`, `https://scooters.taxify.eu`
  - Não é feita a verificação dos *hosts* associados. (`autoverify=true`)
  - A ação é do tipo `VIEW`, esta é utilizada para mostrar dados.
  - As categorias são o *BROWSABLE* e `DEFAULT`.
  - Não é possível obter o ficheiro Digital Asset Links (DAL), não existe `*/.well-known/assetlinks.json`.
  - Falhou a verificação do DAL (`https://digitalassetlinks.googleapis.com/v1/statements:list?source.web.site=https://scooters.taxify.eu`)
- URI: `https://scooters.taxify.eu/qr`, `bolt://action`, `boltprelive://action`, `taxify://action`
  - Não é feita a verificação dos *hosts* associados. (`autoverify=true`)
  - A ação é do tipo `NDEF_DISCOVERED`, é despoletada quando um dispositivo NFC é associado.
  - É utilizada a categoria por omissão (`DEFAULT`).

9 Sintaxe testar deep link, Android Doc. - Disponível em: `https://developer.android.com/training/app-links/deep-linking?hl=pt-br#testing-filters` - Acedido a 5 Out. 2023

  - Não existe nenhum ficheiro DAL associado aos *hosts* anteriores.
  - Não existe o DAL - ou não foi possível identificar o mesmo.

- URI's: `bolt://`, `taxify://`, `geo://`
  - Não é feita a verificação dos *hosts* associados. (`autoverify=true`)
  - A ação é do tipo `VIEW`, esta é utilizada para mostrar dados.
  - As categorias são o *BROWSABLE* e `DEFAULT`.
  - Não é possível obter o ficheiro DAL, não existe `*/.well-known/assetlinks.json`.

- URI: `https://maps.google.com`, `http://maps.google.com`
  - Não é feita a verificação dos *hosts* associados. (`autoverify=true`)
  - A ação é do tipo `VIEW`, esta é utilizada para mostrar dados.
  - As categorias são o *BROWSABLE* e `DEFAULT`.
  - Existe o DAL, no entanto não está associado ao *package* da *Bolt* (*package name*: ee.mtakso.client). Como tal esta verificação foi invalidada.(`https://maps.google.com/.well-known/assetlinks.json` e `http://maps.google.com/.well-known/assetlinks.json`).

- URI: `https://bolt.onelink.me`
  - É feita a verificação do *host* associado (contém `autoverify=true`)
  - A ação é do tipo `VIEW`, esta é utilizada para mostrar dados.
  - As categorias são o *BROWSABLE* e `DEFAULT`.
  - Foi possível obter o ficheiro DAL, disponível em `https://bolt.onelink.me/.well-known/assetlinks.json`

Para testar os *deep links* anteriores de forma dinâmica foi utilizado o *script Frida*, apresentado na listagem 40, obtido através do *codeshare*[10]. Para que possamos observar o resultado da listagem 40 é necessário executar os *deep links*, em conjunto com o *script frida*. Para esse efeito recorremos ao formato apresentado na listagem 39, onde indicamos a ação tal como foi declarada no `manifest`, em conjunto com o *deep link*.

10 Codeshare, Android Deep Link Observer - Disponível em: `https://codeshare.frida.re/@leolashkevych/android-deep-link-observer/` - Acedido a 8 Ago. 2023

Listagem 40: Script Frida para observação dos parâmetros do deeplink

```
Java.perform(function() {
 var Intent = Java.use("android.content.Intent");
 var JavaThread = Java.use("java.lang.Thread");

 Intent.getData.implementation = function() {
        var action = this.getAction() !== null ? this.getAction().toString() :
        ↪  false;
        if (action) {
         console.log("[*] Intent.getData() was called");
         console.log("[*] Activity: " + this.getComponent().getClassName());
         console.log("[*] Action: " + action);
          var uri = this.getData();
          if (uri !== null) {
                console.log("\n[*] Data");
                uri.getScheme() && console.log("- Scheme:\t" + uri.getScheme() +
                ↪  "://");
                uri.getHost() && console.log("- Host:\t\t/" + uri.getHost());
                uri.getQuery() && console.log("- Params:\t" + uri.getQuery());
                uri.getFragment() && console.log("- Fragment:\t" +
                ↪  uri.getFragment());
                console.log("\n\n");
                var th = Java.cast( JavaThread.currentThread(), JavaThread);
                var stack = th.getStackTrace(), e=null;
                console.log("\n[*] Stacktrace");
                for(var i=0; i<stack.length; i++){
                        console.log("\t"+stack[i].getClassName()+"."+stack[i].ge
                        ↪  tMethodName()+"("+stack[i].getFileName()+")");
                }
          } else {console.log("[-] No data supplied.");}
 }return this.getData();        }
});
```

7.4 ANÁLISE FRIDA

Esta secção descreve a metodologia utilizada, bem como os resultados obtidos aos testes efetuados, recorrendo à ferramenta de instrumentalização dinâmica `Frida`.

É necessário consultar toda a documentação oficial[11] para ficar familiarizado com as nuances necessárias para uso da ferramenta. Esta documentação pode ser bastante extensa e complexa, sendo que esta tese apenas se foca nas capacidades da ferramenta para o SO *Android*, fazendo uso da API *JavaScript*. É importante acompanhar a leitura da documentação com a criação de pequenos exemplos práticos para melhor perceber o que está a ser realizado pela ferramenta e para termos um resultado prático das suas capacidades.

É importante sempre que ligar ou reiniciar o DM, iniciar o agente *frida* com permissões de *root*. O comando requerido para efeito encontra-se na Listagem 41.

Listagem 41: Iniciar frida server no dispositivo móvel

```
1
2 $ adb -s 424ac404 shell
3 #Privilegios de root
4 $ su
5 #iniciar o frida-server em segundo plano
6 $ /data/local/tmp/frida-server &
```

A especificação do momento de execução do nosso *script* é feita utilizando as `flags` (`-pause`, ou `-no-pause`), em conjunto com o caminho do *script*. Estas opções permitem-nos determinar o momento no qual pretendemos manipular a aplicação. Esta opção é por vezes necessária em casos onde existe mecanismos de validação como privilégios de root, ou condições que têm que existir por forma a permitir ou não a inicialização da aplicação.

Se pretendermos executar o nosso *script* antes da execução da `main thread` ao iniciar a aplicação, utilizamos a forma apresentada na listagem 42. O comando apresentado na listagem 42 começa por referir o `ID` do dispositivo para o qual pretendemos efetuar a ligação, seguida da *flag* -f que indica, que pretendemos iniciar uma nova instância da aplicação, seguida do seu *package name*, tal como aparece no ficheiro `AndroidManifest.xml`. Se quisermos utilizar a aplicação em execução, utilizamos a *flag* -F, para manipular a aplicação em *Foreground*. Desde a versão `Frida` 15.2 não é necessário usar a *flag* `-no-pause`[12], e por conseguinte se

11 Frida Docs: https://frida.re/docs/javascript-api/ - Acedido a 18 de Maio de 2023
12 Github Frida: https://github.com/charles2gan/GDA-android-reversing-Tool/issues/110 - Acedido a 21 de junho de 2023

pretendermos executar o nosso *script* antes da `main thread` ser iniciada é necessário utilizar a *flag* `-pause`.

Listagem 42: Iniciar instrumentalização da aplicação

```
$ frida -D <ID dispositivo> -f <package name> --pause
```

O apêndice C.1 (página 201) documenta um caso de uso para familiarizar o leitor com as capacidades do *Frida*, recorrendo a uma aplicação menos complexa do que a aplicação *Bolt*.

CONTORNAR O SSL PINNING DA APLICAÇÃO Em grande parte das aplicações que utilizamos, salvo algumas exceções tais como aplicações bancárias, utilizam técnicas de *SSL Pinning* genéricas e bem conhecidas. Como tal existem *scripts* para `Frida` que podem ser utilizados diretamente sobre as aplicações.

O *bypass* ao *SSL Pinning* da aplicação, pode também ser efetuado sem ter localmente o *script*, da forma apresentada na listagem 43. Este caso pode aplicar-se se a máquina do analista não dispor do *script* e rapidamente podemos aceder a um *script* que realiza o *bypass* aos métodos de SSL Pinning mais comuns utilizados por diversas aplicações. Estes *scripts* são partilhados por outros utilizadores[13], é importante comparar o *fingerprint* com o original se não conhecer previamente o mesmo, e/ou consultar o código antes de o executar.

Listagem 43: Bypass SSL Pinning

```

$ frida --codeshare akabe1/frida-multiple-unpinning -U -f ee.mtakso.client
```

FRIDA: BOLT A *Bolt* utiliza uma implementação da interface *TrustManagerImpl*, a qual é utilizada para validar os certificados digitais empregues durante uma comunicação remota. A verificação do certificado confirma se este foi emitido por uma CA de confiança. É sobre os métodos existentes nessa classe que o `Frida` irá ignorar as validações realizadas, considerando todos os certificados como confiáveis.

FRIDA TRACE Esta é uma das funcionalidades pela qual o `Frida` se destaca, e o seu correto uso facilita em grande parte o processo de análise de uma aplicação numa fase inicial. Podemos ver as capacidades da ferramenta através do comando empregue na Listagem 44.

13 Codeshare: https://codeshare.frida.re/browse, data de acesso 18 Maio de 2023

Listagem 44: Obter todas as classes que tenham método onClick

```
1
2         $ frida-trace -U -F -j '*!onClick'
```

OBTER DADOS JSON EM TEMPO REAL Ao efetuar *hook* ao *package* da aplicação que é referenciado para manipulação de dados em *JSON*, ilustrado na Listagem 45. O *script* de *SSL Pinning* têm de ser iniciado previamente, se tal não for efetuado os dados *JSON* irão aparecer sobre o formato cifrado.

Listagem 45: Listagem em tempo real da utilização JSON pela Aplicação

```
1
2 $ frida-trace -U -f ee.mtakso.client -j 'org.json.JSONObject!*'
```

FRIDA TRACE - CONSULTAR ACESSOS À BASE DE DADOS O `Frida` permite listar todas as funções que são utilizadas por uma determinada biblioteca referenciada pela aplicação. No caso específico sabemos que a aplicação faz uso do sistema de BD *SQLite* para escrever, ler e executar *querys* às BD locais. O resultado ao comando apresentado na Listagem 46, identifica as bibliotecas que contêm as funções utilizadas pela aplicação *Bolt*.

Listagem 46: Listar as funções para cada uma das bibliotecas que fazem uso do sqlite

```
1
2                 $ frida-trace -U -i open* -I *sqlite* -f ee.mtakso.client
```

Utilizamos o resultado obtido da Listagem 46 para identificar as classes mais relevantes a explorar no código fonte. Por sua vez se estas utilizarem bibliotecas externas, a sua análise pode ser efetuada por ferramentas como o *Ghidra* ou o *IDA pro*.

Após consultarmos de que forma são realizados os acessos às base de dados locais, demos por encerrada a análise dinâmica da aplicação *Bolt*.

8

CONCLUSÕES

Ao instalar-se uma aplicação móvel no seu *smartphone*, aceitando a respetiva política de utilização, o utilizador está a confiar na robustez e idoneidade da aplicação. A robustez pode ser fragilizada pela existência de falhas de segurança na aplicação que podem comprometer todo o conteúdo do dispositivo, ao passo que a política de utilização, usualmente composta por várias páginas de texto com forte conteúdo jurídico, pode esconder o uso de dados do utilizador, sem que o mesmo tenha plena consciência da situação.

Este trabalho procurou estudar as metodologias para análise forense e de segurança a aplicações móveis *Android*, recorrendo exclusivamente a ferramentas de código aberto. O objetivo subjacente foi o de compreender os desafios e oportunidades associados à proteção da segurança na plataforma *Android*, fornecendo uma base sólida para investigar questões de segurança emergentes e estratégias de mitigação. Assim, ao longo do trabalho, foram exploradas uma variedade de conceitos, tecnologias e técnicas de análise estática e dinâmica, abrangendo desde a arquitetura do sistema *Android* até à segurança de rede e análise forense de artefactos em dispositivos móveis. Destaca-se a importância da compreensão dos componentes do sistema operativos *Android* e das muitas ferramentas disponíveis para avaliar a segurança de aplicações *Android*, identificar vulnerabilidades e potenciais ameaças. Este estudo compilou uma variedade de informações que estão dispersas em várias fontes de literatura e documentação, procurando sempre uma abordagem prática, documentando as metodologias, ferramentas e modos de uso.

As ferramentas que foram utilizadas para realizar este trabalho são *open-source*, com objetivos e limitações intrínsecas às suas próprias características. Com isto queremos dizer que apresentam limitação na quantidade e qualidade de informação criada. A utilização apropriada das metodologias e ferramentas exige um estudo prévio, para que seja possível com base nos resultados obtidos adaptar o seu uso e, muitas vezes, requer uma abordagem de tentativa e erro para conseguir obter resultados concretos. Uma característica que é comum no ramo de segurança de aplicações, é que na grande maioria dos casos nenhuma aplicação ou ferramenta vai identificar de forma correta as falhas que existem, gerando, ao invés, um elevado

número de falsos positivos. Por sua vez, durante uma análise dinâmica é expectável que seja necessário forçar e testar a aplicação com diversas técnicas e ferramentas, procurando contornar os mecanismos de defesas das aplicações.

Neste trabalho, para caso de estudo, foi selecionada a aplicação '*Bolt*', dado que a mesma processa dados sensíveis dos utilizadores, desde dados de identificação pessoal, localização e meios de pagamento. Do estudo efetuado, foi percetível que a aplicação *Bolt* recolhe informação para além do considerado necessário. O excesso de informação recolhida afeta tanto os utilizadores, como o *hardware* incluindo informação de rede e localização, a qual em muitos casos é não necessária para a função ou depuração da aplicação. Este trabalho refere com detalhe os diretórios e artefactos com maior valor forense que são recolhidos pela aplicação *Bolt*. Estes artefactos podem ser relevantes, por exemplo, para forças policiais ou judiciais no decurso das respetivas investigações. Os artefactos podem também ser importantes para os utilizadores da aplicação Bolt terem conhecimento dos dados recolhidos pela aplicação, assim como dos *trackers* existentes e a informação que é recolhida sem que sejam alertados devidamente para o mesmo.

O autor deste trabalho teve que enfrentar, na primeira pessoa, toda a diversidade, variedade e subtilezas do sistema operativo *Android*, das metodologias e ferramentas disponíveis para análise de segurança e forense de aplicações móveis. Considera que o presente documento pode contribuir para tornar mais acessível a prática de análise de segurança e de informática forense a aplicações *Android*. Concretamente, este documento pode ser empregue para futuros investigadores de segurança de aplicações móveis que estejam a iniciar a sua jornada, procurando colmatar a escassez de trabalhos que detalhem o procedimento utilizado, que apresenta um conjunto de ferramentas que têm provas dadas e que são aceites como robustas nas suas funções.

8.1 TRABALHO FUTURO

Concluímos esta dissertação apresentando de seguida diversas direções que podem ser tomadas de futuro.

O maior objetivo como trabalho futuro seria continuar a análise da aplicação de forma dinâmica. Em específico, considera-se pertinente uma análise recorrendo às ferramentas *Frida* e *radare2* da aplicação *Bolt*. Ambas as ferramentas apresentam elevadas capacidades, embora tenham uma acentuada curva de aprendizagem, requerendo um elevado investimento para domínio das funcionalidades mais avançadas.

Importa notar que o uso do *Frida* com o *BurpSuite* revelou ser uma combinação muito interessante durante a análise dinâmica de uma aplicação de *ride-hailing* como é o caso da aplicação *Bolt - Share a Ride*'.

Outro possível trabalho futuro é uma análise mais aprofundada às API empregues pela aplicação *Bolt*. A análise de API empregues é um dos tópicos mais interessantes para criar resultados objetivos. Complementarmente, é sugerido para trabalho futuro o uso de toda a bateria de testes desenvolvida pela OWASP. No seguimento desse mesmo tópico seria interessante a existência de uma ferramenta de automação do processo de verificação do MASTG. Note-se contudo que o desenvolvimento de tal ferramenta seria uma tarefa muito exigente, pois tal como referido anteriormente, os resultados deste tipo de ferramentas automatizadas, apresentam muitos falsos positivos.

BIBLIOGRAFIA

Amalfitano, Domenico et al. (set. de 2015). «MobiGUITAR: Automated Model-Based Testing of Mobile Apps». Em: *IEEE Software* 32.5, pp. 53–59. DOI: 10.1109/ms.2014.55.

Anmol Misra, Abhishek Dubey (abr. de 2016). *Android Security*. Taylor & Francis Ltd. 280 pp. ISBN: 143989647X. URL: https://www.ebook.de/de/product/21178347/anmol_misra_abhishek_dubey_android_security.html.

Arzt, S et al. (2014). «FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps». Em: *Proc. of PLDI*, pp. 259–269.

Au, Kathy et al. (2012). «PScout». Em: *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press, pp. 217–228. DOI: 10.1145/2382196.2382222.

Avdiienko, Vitalii et al. (mai. de 2015). «Mining Apps for Abnormal Usage of Sensitive Data». Em: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE, pp. 426–436. DOI: 10.1109/icse.2015.61.

Babil, G et al. (jul. de 2013). «On the effectiveness of dynamic taint analysis for protecting against private information leaks on android-based devices». Em: *2013 International Conference on Security and Cryptography (SECRYPT)*, pp. 1–8.

Barmpatsalou, Konstantia, Tiago Cruz et al. (mai. de 2018). «Current and Future Trends in Mobile Device Forensics: A Survey». Em: *ACM Comput. Surv.* 51.3. ISSN: 0360-0300. DOI: 10.1145/3177847. URL: https://doi.org/10.1145/3177847.

Barmpatsalou, Konstantia, Dimitrios Damopoulos et al. (2013). «A critical review of 7 years of Mobile Device Forensics». Em: *Digital Investigation* 10.4, pp. 323–349. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2013.10.003. URL: https://www.sciencedirect.com/science/article/pii/S1742287613001096.

Bhoraskar, R et al. (2014). «Brahmastra: Driving Apps to Test the Security of Third-Party Components». Em: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, pp. 1021–1036.

Cao, Yinzhi et al. (2015). «EdgeMiner: Automatically Detecting Implicit Control Flow Transitions through the Android Framework». Em: *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society. DOI: 10.14722/ndss.2015.23140.

Casey, Eoghan (2011). *Digital evidence and computer crime forensic science, computers and the Internet. forensic science, computers and the Internet*. Academic Press. ISBN: 9780123742681.

Casino, Fran et al. (2022). «Research Trends, Challenges, and Emerging Topics in Digital Forensics: A Review of Reviews». Em: *IEEE Access* 10. Cited by: 3; All Open Access, Gold Open Access, Green Open Access, pp. 25464–25493. DOI: 10.1109/ACCESS.2022.3154059.

Chantzis, Fotios et al. (2020). *Practical IoT Hacking*. No Starch Press, Incorporated, p. 460. ISBN: 978-1-7185-0090-7.

Chau, Ngoc-Tu e Souhwan Jung (2018). «Dynamic analysis with Android container: Challenges and opportunities». Em: *Digital Investigation* 27, pp. 38–46. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2018.09.007. URL: https://www.sciencedirect.com/science/article/pii/S1742287618301257.

Chen, Ping et al. (2016). «Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware». Em: *ICT Systems Security and Privacy Protection*. Ed. por Jaap-Henk Hoepman e Stefan Katzenbeisser. Cham: Springer International Publishing, pp. 323–336. ISBN: 978-3-319-33630-5.

Chen, Shuhui et al. (2016). «Investigating and Revealing Privacy Leaks in Mobile Application Traffic». Em: *2019 Wireless Days (WD)*. IEEE, pp. 361–374. DOI: 10.1109/wd.2019.8734246.

Chess, B e G Mcgraw (nov. de 2004). «Static analysis for security». Em: *IEEE Security and Privacy Magazine* 2.6, pp. 76–79. DOI: 10.1109/msp.2004.111.

Christodorescu, M et al. (2005). «Semantics-aware malware detection». Em: *2005 IEEE Symposium on Security and Privacy*. IEEE, pp. 32–46. DOI: 10.1109/sp.2005.20.

Christodorescu, Mihai e Somesh Jha (1 de jan. de 2006). *Static Analysis of Executables to Detect Malicious Patterns*. Rel. téc. DOI: 10.21236/ada449067.

Conlan, Kevin, Ibrahim Baggili e Frank Breitinger (2016). «Anti-forensics: Furthering digital forensic science through a new extended, granular taxonomy». Em: *Digital Investigation* 18, S66–S75. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2016.04.006. URL: https://www.sciencedirect.com/science/article/pii/S1742287616300378.

Continella, Andrea et al. (2017). «Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis». Em: *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, pp. 1–16. DOI: 10.14722/ndss.2017.23465.

Cybersecurity, Lab of e Digital Forensics at IPLeiria (fev. de 2022). «AndroidStudioEmulator-acquireAppsData».

EIA, phoyo (jan. de 2022). *Estonian unicorn Bolt raises €628M, reaches €7.4B valuation Estonia Unicorn*. URL: https://investinestonia.com/estonian-unicorn-bolt-raises-e628m-reaches-e7-4b-valuation/ (acedido em 26/10/2022).

Elenkov, Nikolay (2015). *Android Securit Internals: An In-Depth Guide toAndroid's SecurityArchitecture*. Ed. por no starch press. William Pollock. ISBN: 978-1-59327-581-5.

Fahl, Sascha et al. (2012). «Why eve and mallory love android». Em: *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press, pp. 50–61. DOI: 10.1145/2382196.2382205.

Farjamfar, Anahita e Mohd Taufik Abdullah (jul. de 2014). «A Review on Mobile Device's Digital Forensic Process Models». Em: *Research Journal of Applied Sciences, Engineering and Technology* 8, pp. 358–366.

Faruki, Parvez et al. (set. de 2014). «Evaluation of Android Anti-malware Techniques against Dalvik Bytecode Obfuscation». Em: *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, pp. 414–421. DOI: 10.1109/trustcom.2014.54.

Georgiev, Martin et al. (2012). «The most dangerous code in the world». Em: *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press, pp. 38–49. DOI: 10.1145/2382196.2382204.

Gibler, Clint et al. (2012). «AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale». Em: *Trust and Trustworthy Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 291–307. DOI: 10.1007/978-3-642-30921-2_17.

Guibernau, Francis (jan. de 2020). «Catch Me If You Can!—Detecting Sandbox Evasion Techniques». Em: San Francisco, CA: USENIX Association.

Gupta, Sandeep, Attaullah Buriro e Bruno Crispo (2019). «DriverAuth: A risk-based multi-modal biometric-based driver authentication scheme for ride-sharing platforms». Em: *Computers & Security* 83, pp. 122–139. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0167404818310113.

Harris, Ryan (2006). «Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem». Em: *Digital Investigation* 3. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06), pp. 44–49. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2006.06.005. URL: https://www.sciencedirect.com/science/article/pii/S1742287606000673.

Horsman, Graeme (2017). «Can we continue to effectively police digital crime?» Em: *Science & Justice* 57.6, pp. 448–454. ISSN: 1355-0306. DOI: https://doi.org/10.1016/j.scijus.2017.06.001. URL: https://www.sciencedirect.com/science/article/pii/S1355030617300801.

Jacobs, Yasmine (mar. de 2022). *This is what Bolt says it does after a sexual assault case is opened*. https://www.iol.co.za/news/south-africa/western-cape/this-is-what-bolt-says-it-does-after-a-sexual-assault-case-is-opened-cd8eaae9-d30c-42b6-8f61-ad7bfd6e4234.

Kessler, Gary C. (nov. de 2015). «Are mobile device examinations practiced like 'forensics'?» Em: *Digital Evidence and Electronic Signature Law Review* 12.0. DOI: 10.14296/deeslr.v12i0.2237.

Kim, Dohyun e Sangjin Lee (2020). «Study of identifying and managing the potential evidence for effective Android forensics». Em: *Forensic Science International: Digital Investigation* 33, p. 200897. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2019.200897. URL: https://www.sciencedirect.com/science/article/pii/S1742287619301367.

Le, Anh et al. (17 de ago. de 2015). «AntMonitor». Em: *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*. ACM, pp. 15–20. DOI: 10.1145/2787394.2787396.

Lindorfer, Martina et al. (set. de 2014). «ANDRUBIS – 1,000,000 Apps Later: A View on Current Android Malware Behaviors». Em: *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. IEEE, pp. 3–17. DOI: 10.1109/badgers.2014.7.

Liu, Minxing et al. (23 de fev. de 2016). «Identifying and Analyzing the Privacy of Apps for Kids». Em: *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*. ACM. DOI: 10.1145/2873587.2873597.

Maiorca, Davide et al. (jun. de 2015). «Stealth attacks: An extended insight into the obfuscation effects on Android malware». Em: *Computers & Security* 51, pp. 16–31. DOI: 10.1016/j.cose.2015.02.007.

Matulis, Nina e Umit Karabiyik (2022). «Digital Forensics for Mobility as A Service Platform: Analysis of Uber Application on iPhone and Cloud». Em: *Annual*

ADFSL Conference on Digital Forensics, Security and Law. 5. URL: https://commons.erau.edu/adfsl/2022/presentations/5.

Min, Zhao et al. (mar. de 2019). «Android software vulnerability mining framework based on dynamic taint analysis technology». Em: *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, pp. 2112–2115. DOI: 10.1109/itnec.2019.8729217.

Moreb, Mohammed (2022). *Practical Forensic Analysis of Artifacts on IOS and Android Devices. Investigating Complex Mobile Devices*. Apress L. P. ISBN: 9781484280256.

ÖZGÜR, Berkecan et al. (2021). «A Suggested Model for Mobile Application Penetration Test Framework». Em: *2021 International Conference on Information Security and Cryptology (ISCTURKEY)*, pp. 18–21. DOI: 10.1109/ISCTURKEY53027.2021.9654417.

Qureshi, Muhammad Ali e El-Sayed M El-Alfy (2019). «Bibliography of digital image anti-forensics and anti-anti-forensics techniques». Em: *IET Image Processing* 13.11, pp. 1811–1823. DOI: https://doi.org/10.1049/iet-ipr.2018.6587.

Rao, Ashwin et al. (2012). «Meddle». Em: *Proceedings of the 2012 ACM conference on CoNEXT student workshop - CoNEXT Student '12*. ACM Press, pp. 65–66. DOI: 10.1145/2413247.2413286.

Ratazzi, Paul (dez. de 2016). «Understanding and Improving Security of the Android Operating System». Tese de doutoramento.

Razaghpanah, Abbas, Arian Niaki et al. (2017). «Studying TLS Usage in Android Apps». Em: *Proceedings of the Applied Networking Research Workshop*. ACM, pp. 350–362. DOI: 10.1145/3232755.3232779.

Razaghpanah, Abbas, Narseo Vallina-Rodriguez et al. (out. de 2015). «Haystack: In Situ Mobile Traffic Analysis in User Space». Em.

Reddy, Niranjan (2019). «Mobile Forensics». Em: *Practical Cyber Forensics: An Incident-Based Approach to Forensic Investigations*. Berkeley, CA: Apress, pp. 205–239. ISBN: 978-1-4842-4460-9. DOI: 10.1007/978-1-4842-4460-9_7. URL: https://doi.org/10.1007/978-1-4842-4460-9_7.

Sarwar, G et al. (2013). «On the effectiveness of dynamic taint analysis for protecting against private information leaks on androidbased devices». Em: *SECRYPT*. Vol. 96435.

Schwartz, Edward, Thanassis Avgerinos e David Brumley (2010). «All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask)». Em: *2010 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE, pp. 317–331. DOI: 10.1109/sp.2010.26.

Skulkin, Oleg, Donnie Tindall e Rohit Tamma (2018). *Learning Android Forensics. Analyze Android Devices with the Latest Forensic Tools and Techniques, 2nd Edition*. Packt Publishing, Limited, p. 328. ISBN: 978-1-78913-101-7.

Song, Yihang e Urs Hengartner (12 de out. de 2015). «PrivacyGuard». Em: *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM. DOI: 10.1145/2808117.2808120.

Talegaon, Samir e Ram Krishnan (nov. de 2020). «A Formal Specification of Access Control in Android with URI Permissions». Em: *Information Systems Frontiers* 23.4, pp. 849–866. DOI: 10.1007/s10796-020-10066-9.

Tallinn (jan. de 2020). *€50 million in financing for Estonian ride-hailing platform Bolt Press release*. URL: https://ec.europa.eu/commission/presscorner/detail/en/ip_20_74 (acedido em 26/10/2022).

Tamma, Rohit et al. (2018). *Practical Mobile Forensics - Third Edition: A hands-on guide to mastering mobile forensics for the iOS, Android, and the Windows Phone platforms*. Packt Publishing, p. 402. ISBN: 9781788839198.

— (2020). *Practical Mobile Forensics. Forensically Investigate and Analyze IOS, Android, and Windows 10 Devices, 4th Edition*. Packt Publishing, Limited, p. 400. ISBN: 9781838647520.

Umer Farooq (2018). «Android Operating System Architecture». en. Em: DOI: 10.13140/RG.2.2.20829.72169.

Votipka, Daniel, Timothy Vidas e Nicolas Christin (2013). «Passe-Partout: A General Collection Methodology for Android Devices». Em: *IEEE Transactions on Information Forensics and Security* 8.12, pp. 1937–1946. DOI: 10.1109/TIFS.2013.2285360.

Wang, Hai e Hai Yang (2019). «Ridesourcing systems: A framework and review». Em: *Transportation Research Part B: Methodological* 129, pp. 122–155. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2019.07.009. URL: https://www.sciencedirect.com/science/article/pii/S019126151831172X.

Yaghmour, Karim (2012). *Embedded Android. porting, extending, and customizing*. O'Reilly Media, p. 412. ISBN: 9781449308292.

Yang, Yufei et al. (2013). «Execution Enhanced Static Detection of Android Privacy Leakage Hidden by Dynamic Class Loading». Em: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, pp. 1043–1054. DOI: 10.1109/compsac.2019.00029.

Zhang, Xiaolu et al. (2021). «Android application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations». Em: *Forensic Science International: Digital Investigation* 39, p. 301285. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2021.301285.

URL: https://www.sciencedirect.com/science/article/pii/S2666281721002031.

Zhao, Qingchuan et al. (2019). «Geo-locating Drivers: A Study of Sensitive Data Leakage in Ride-Hailing Services». Em: *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society. DOI: 10.14722/ndss.2019.23052.

Zimmeck, Sebastian et al. (2017). «Automated Analysis of Privacy Requirements for Mobile Apps». Em: *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society. DOI: 10.14722/ndss.2017.23034.

APÊNDICES

APÊNDICE A

A.1 ESTRUTURA DE DIRETÓRIOS DE BACKUP

De seguida é apresentada a estrutura do diretório obtido através do backup da aplicação, para *Android* 6.0.

Figura 42: Estrutura Backup Parte(1/3)

Figura 43: Estrutura Backup Parte(2/3)

Figura 44: Estrutura Backup Parte(3/3)

A.2 ESTRUTURA DE DIRETÓRIOS LOCAIS - BOLT

Listagem 47: Estrutura de diretórios do DM após interação com a aplicação Bolt

```
.
|-data
|---user
|-----0
|-------ee.mtakso.client
|---------app_textures
|---------app_webview
|-----------Default
|-------------blob_storage
|---------------3c5f25a1-bff1-43da-8361-998e6663993a
|-------------Local Storage
|---------------leveldb
|-------------Session Storage
|---------cache
|-----------com.google.android.gms.maps.volley
|-----------instrument
|-----------lottie_network_cache
|-----------oat
|-------------arm64
|-----------WebView
|-------------Crashpad
|---------------attachments
|---------------completed
|---------------new
|---------------pending
|-------------Crash Reports
|-------------Default
|---------------HTTP Cache
|-----------------Cache_Data
|-------------------index-dir
|-----------------Code Cache
|-------------------js
|---------------------index-dir
|-------------------wasm
|---------------------index-dir
|-------------SafeBrowsing
|---------code_cache
|---------databases
|---------files
|-----------AFRequestCache
|-----------phenotype
|-------------shared
|-----------sinch
|-------------db
|---------------950A17B1
|-----------splitcompat
|-------------2086
|---------------verified-splits
|---------no_backup
|---------shared_prefs
|---user_de
|-----0
|-------ee.mtakso.client
|---------cache
|---------code_cache
|---------databases
|---------files
|---------no_backup
|---------shared_prefs
```

A.3 RESULTADOS DA EXECUÇÃO DE FERRAMENTAS DE ANÁLISE

A.3.1 *DroidLysis*

Os resultados aqui demonstrados resultam da execução da ferramenta *droidlysis*. Uma explanação das características e objetivos no uso da ferramenta encontram-se expostos em 4.2.11.

Listagem 48: Relatório DroidLysis3.py (parte 1/5)

```
$ python3 droidlysis3.py --enable-procyon -i Bolt52.1.apk -o teste1
Processing: /Dissertacao/Apk original/Bolt52.1.apk ...
Filename: /Dissertacao/Apk original/Bolt52.1.apk
Launching unzip process on /Dissertacao/Apk original/Bolt52.1.apk with output dir=teste1/Bolt52.1.apk-3bc581327e5353830b03dba87efe9fa60afce9e774a027488edd345bda9e36fe/unzipped
caution: filename not matched:  classes9.dex
============= Report ============
Sanitized basename  : Bolt52.1.apk
SHA256              : 3bc581327e5353830b03dba87efe9fa60afce9e774a027488edd345bda9e36fe
File size           : 64590299 bytes
Is small            : False
Nb of classes       : 45605
Nb of dirs          : 3362
Certificate properties
algo                : SHA1withRSA(weak)
serialno            : 51c97f6a
country             : 500
owner               : O=MtaksoOÜ
timestamp           : (1981, 1, 1, 1, 1, 2)
year                : 1981

Manifest properties
activities          : ["'ee.mtakso.client.newbase.RideHailingMapActivity'",
"'ee.mtakso.client.newbase.deeplink.DeeplinkActivity'",
"'ee.mtakso.client.newbase.deeplink.AppsFlyerDeeplinkActivity'",
"'ee.mtakso.client.activity.ThreeDSActivity'",
"'ee.mtakso.client.activity.voip.VoipCallActivity'",
"'com.facebook.FacebookActivity'",
"'eu.bolt.verification.sdk.VerificationSDKActivity'",
"'com.clevertap.android.sdk.InAppNotificationActivity'",
"'com.clevertap.android.sdk.inbox.CTInboxActivity'",
"'co.paystack.android.ui.PinActivity'",
"'co.paystack.android.ui.OtpActivity'",
"'co.paystack.android.ui.AuthActivity'",
"'co.paystack.android.ui.CardActivity'",
"'co.paystack.android.ui.AddressVerificationActivity'",
"'com.survicate.surveys.SurveyActivity'",
"'com.google.android.gms.auth.api.signin.internal.SignInHubActivity'",
"'com.google.android.gms.common.api.GoogleApiActivity'",
"'com.facebook.notifications.internal.activity.CardActivity'",
"'com.adyen.threeds2.internal.ui.activity.ChallengeActivity'",
"'com.facebook.CustomTabMainActivity'", "'com.facebook.CustomTabActivity'",
"'com.patloew.rxlocation.LocationSettingsActivity'", "'com.google.android.play.core.missingsplits.PlayCoreMissingSplitsActivity'",
"'com.google.android.play.core.common.PlayCoreDialogWrapperActivity'",
"'com.veriff.sdk.views.base.verification.VeriffActivity'"]
libraries           : ["'org.apache.http.legacy'",
"'androidx.window.extensions'", "'androidx.window.sidecar'"]
minSDK              : 21
package_name        : ee.mtakso.client
permissions         : ['INTERNET', 'FOREGROUND_SERVICE',
'ACCESS_COARSE_LOCATION', 'ACCESS_FINE_LOCATION', 'ACCESS_WIFI_STATE',
'ACCESS_NETWORK_STATE', 'CAMERA', 'GET_PACKAGE_SIZE', 'VIBRATE',
'com.google.android.gms.permission.AD_ID', 'NFC', 'POST_NOTIFICATIONS',
'WAKE_LOCK', 'com.google.android.c2dm.permission.RECEIVE',
'com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE',
'RECEIVE_BOOT_COMPLETED', 'RECORD_AUDIO', 'MODIFY_AUDIO_SETTINGS',
'BLUETOOTH', 'USE_FULL_SCREEN_INTENT']
```

Listagem 49: Relatório DroidLysis3.py (parte 2/5)

```
providers            :
["'com.facebook.FacebookContentProvider'",
 ↪  "'androidx.core.content.FileProvider'",
 ↪  "'com.google.mlkit.common.internal.MlKitInitProvider'",
 ↪  "'com.google.firebase.perf.provider.FirebasePerfProvider'",
 ↪  "'com.google.firebase.provider.FirebaseInitProvider'",
 ↪  "'com.facebook.internal.FacebookInitProvider'",
 ↪  "'androidx.startup.InitializationProvider'"]

receivers            : ["'com.appsflyer.MultipleInstallBroadcastReceiver'",
 ↪  "'eu.bolt.client.chat.notifications.NotificationDeleteIntentReceiver'",
 ↪  "'com.clevertap.android.sdk.pushnotification.CTPushNotificationReceiver'",
 ↪  "'com.google.firebase.iid.FirebaseInstanceIdReceiver'",
 ↪  "'com.adyen.threeds2.internal.AppUpgradeBroadcastReceiver'",
 ↪  "'com.google.android.gms.measurement.AppMeasurementReceiver'",
 ↪  "'com.facebook.CurrentAccessTokenExpirationBroadcastReceiver'",
 ↪  "'androidx.work.impl.utils.ForceStopRunnable$BroadcastReceiver'", "'androidx
 ↪  .work.impl.background.systemalarm.ConstraintProxy$BatteryChargingProxy'",
 ↪  "'androidx.work.impl.background.systemalarm.ConstraintProxy$BatteryNotLowPro
 ↪  xy'",
 ↪  "'androidx.work.impl.background.systemalarm.ConstraintProxy$StorageNotLowPro
 ↪  xy'",
 ↪  "'androidx.work.impl.background.systemalarm.ConstraintProxy$NetworkStateProx
 ↪  y'", "'androidx.work.impl.background.systemalarm.RescheduleReceiver'",
 ↪  "'androidx.work.impl.background.systemalarm.ConstraintProxyUpdateReceiver'",
 ↪  "'androidx.work.impl.diagnostics.DiagnosticsReceiver'",
 ↪  "'com.google.android.datatransport.runtime.scheduling.jobscheduling.AlarmMan
 ↪  agerSchedulerBroadcastReceiver'"]

services             : ["'ee.mtakso.client.ribs.root.ridehailing.safetytoolkit.so
 ↪  sintegration.service.IncidentReportingService'",
 ↪  "'ee.mtakso.client.fcm.FcmListenerService'", "'com.clevertap.android.sdk.pus
 ↪  hnotification.amp.CTBackgroundIntentService'",
 ↪  "'com.clevertap.android.sdk.pushnotification.amp.CTBackgroundJobService'",
 ↪  "'com.google.firebase.messaging.FirebaseMessagingService'",
 ↪  "'com.google.firebase.components.ComponentDiscoveryService'",
 ↪  "'com.google.mlkit.common.internal.MlKitComponentDiscoveryService'",
 ↪  "'com.google.android.gms.auth.api.signin.RevocationBoundService'",
 ↪  "'com.google.android.gms.measurement.AppMeasurementService'",
 ↪  "'com.google.android.gms.measurement.AppMeasurementJobService'",
 ↪  "'androidx.work.impl.background.systemalarm.SystemAlarmService'",
 ↪  "'androidx.work.impl.background.systemjob.SystemJobService'",
 ↪  "'androidx.work.impl.foreground.SystemForegroundService'",
 ↪  "'androidx.camera.core.impl.MetadataHolderService'",
 ↪  "'com.veriff.sdk.camera.core.impl.MetadataHolderService'",
 ↪  "'androidx.room.MultiInstanceInvalidationService'", "'com.google.android.dat
 ↪  atransport.runtime.backends.TransportBackendDiscovery'",
 ↪  "'com.google.android.datatransport.runtime.scheduling.jobscheduling.JobInfoS
 ↪  chedulerService'",
 ↪  "'com.google.android.play.core.assetpacks.AssetPackExtractionService'",
 ↪  "'com.google.android.play.core.assetpacks.ExtractionForegroundService'",
 ↪  "'eu.bolt.voip.service.VoipService'",
 ↪  "'com.sinch.android.rtc.internal.client.fcm.InstanceIDTokenService'",
 ↪  "'com.veriff.sdk.service.SendAuthenticationFlowDataToServerService'"]
targetSDK            : 32
```

Listagem 50: Relatório DroidLysis3.py (parte 3/5)

```
    Smali properties /  What the Dalvik code does
accessibility_servic: True (Work with accessibility settings (use, or implement
↪  a service). Meant to help users with disabilities, but often abused by
↪  malicious apps.)
airplane            : True (Detects phone airplane mode)
android_id          : True (Retrieves the Android ID)
base64              : True (Uses Base64 encoder/decoder)
battery             : True (Gets battery info (e.g. how charged, temperature))
board               : True (Retrieves hardware board information)
bootloader          : True (Retrieves version of bootloader)
brand               : True (Retrieves phone brand name)
call                : True (Can place calls)
camera              : True (Uses the phone camera)
check_permission    : True (Checks for given permissions)
class_loader        : True (Get class loader. Can be used for reflexion or
↪  dynamic class loading)
contacts            : True (Reads or lists phone contacts)
cpu_abi             : True (Retreives CPU ABI)
dhcp_server         : True (Queries the address of a DHCP server)
dns                 : True (Queries the address of a DNS server)
email               : True (Reading/writing or sending an email)
encryption          : True (Uses encryption)
execute_native      : True (Executes shell or native executables)
fingerprint         : True (Retrieves hardware Build fingerprint)
get_external_storage: True (Reads storage state, possibly to tell if SD card
↪  mounted read-only or read-write)
get_imei            : True (Retrieves phone IMEI)
get_installed_packag: True (Lists installed packages)
get_installer_packag: True (Gives the name of the app which installed a given
↪  package)
get_line_number     : True (Retrieves end user Phone number (line number))
get_mac             : True (Retrieves MAC address)
get_network_operator: True (Retrieves Network operator)
get_package_info    : True (Gets information on package)
get_sim_country_iso : True (Retrieves SIM country)
get_sim_operator    : True (Retrieves SIM operator)
get_sim_serial_numbe: True (Retrieves SIM serial number)
gps                 : True (Uses GPS location)
hardware            : True (Retrieves phone hardware information)
hide_softkeyboard   : True (Hides software keyboard)
http                : True (Performs HTTP GET)
intent_chooser      : True (Uses intent chooses to ask end-user what application
↪  to use when a given event occurs (e.g which email app to use to send an
↪  email))
ip_address          : True (Retrieves the device IP address)
ip_properties       : True (Gets the netmask or gateway used by the device)
javascript          : True (Loads JavaScript in WebView)
jni                 : True (Uses Java JNI)
json                : True (Uses JSON objects)
link_speed          : True (Gets link speed for Wifi)
load_library        : True (Loads a native library)
manufacturer        : True (Retrieves hardware manufacturer name)
microphone          : True (Mutes the microphone)
model               : True (Retrieves hardware build model)
nop                 : True (DEX bytecode contains NOP instructions.)
package_sig         : True (Reads signatures of packages)
post                : True (Tries to perform an HTTP POST. There might be False
↪  Positives...)
product             : True (Retrieves hardware build product)
record              : True (Records audio on the phone)
record_screen       : True (Records screen)
reflection          : True (Uses Java Reflection)
ringer              : True (Gets or sets ringer mode)
rooting             : True (Searches for or uses applications typically
↪  installed on rooted phones.)
```

Listagem 51: Relatório DroidLysis3.py (parte 4/5)

```
rssi                : True (Gets Wifi RSSI)
scp                 : True (Sends or retrieves files via SCP)
sensor              : True (Lists hardware sensors or receives sensor events.
↪  Sometimes abused to check the phone is running in a sandbox.)
set_component       : True (Might be trying to hide the application icon)
ssh                 : True (Application uses SSH)
ssid                : True (Retrieves SSID used by Wifi)
stacktrace          : True (Get stack traces. Can be used as Anti Frida
↪  technique.)
su                  : True (Uses Su. Perhaps to test if device is rooted.)
uri                 : True (Parses a URL. Will usually just display the URL, but
↪  not post info.)
user_agent          : True (Specifies a HTTP User Agent)
uuid                : True (Creates a random identifier. Used to identify the
↪  user.)
version             : True (Build version)
vibrate             : True (Uses phone vibrations)
webview             : True (Displays a URL in the WebView. Very much used to
↪  display custom pages with JavaScript, sometimes malicious...)
zip                 : True (Zips or unzips files)
multidex            : ['classes2.dex', 'classes3.dex', 'classes4.dex',
↪  'classes5.dex', 'classes6.dex', 'classes7.dex', 'classes8.dex'] (None)

    Wide properties /  What Resources/Assets do
urls                : ['https://netty.io', '0.4.0.127',
↪  'http://www.eclipse.org', 'https://wiki.eclipse.org', 'http://www.ccil.org',
↪  'https://firebase.google.com', 'https://goo.gl', '72.74.71.7',
↪  '1.2.156.101', 'https://node.bolt.eu', 'https://node.prelive.bolt.eu',
↪  'https://user.live.boltsvc.net', 'https://user.prelive.boltsvc.net',
↪  'https://api.mixpanel.com', '1.2.643.2', 'https://respondent.survicate.com',
↪  'https://survey.survicate.com', '1.3.6.1', '1.3.132.0', '1.3.132.1',
↪  'https://taxify.atlassian.net', '2.16.840.1', '2.5.4.3', '2.5.4.6',
↪  '2.5.4.7', '2.5.4.8', '2.5.4.10', '2.5.4.11', '2.5.4.20', '2.5.4.41',
↪  '2.5.4.97', '1.3.14.3', '1.3.36.3', '2.5.8.1', 'https://img.utdstc.com',
↪  'https://i1.wp.com', 'https://lumiere-a.akamaihd.net',
↪  'https://static.wikia.nocookie.net', 'https://upload.wikimedia.org',
↪  'https://starwarsblog.starwars.com', 'https://i.insider.com',
↪  'https://static.onecms.io', 'https://memegenerator.net',
↪  'https://i.imgflip.com', '1.2.840.100', '1.3.133.16', '1.2.250.1',
↪  '1.2.840.113', 'https://bolt.eu', 'https://google.com',
↪  'https://placekitten.com', 'https://habrastorage.org',
↪  'https://api.paymentsos.com', 'https://blog.passmefast.co.uk',
↪  'https://www.politsei.ee', '1.2.643.7', '2.5.1.1', '1.3.14.7',
↪  'https://images.bolt.eu', 'https://media-exp1.licdn.com',
↪  'https://images.unsplash.com',
↪  'https://vereshchaka-private.s3.eu-central-1.amazonaws.com',
↪  'https://alchemy.veriff.com', 'https://static.veriff.com',
↪  'https://www.veriff.com', 'https://magic.veriff.me',
↪  'https://www.coinbase.com', 'https://handshake.probity.io',
↪  'https://admin.taxify.eu']

```

Listagem 52: Relatório DroidLysis3.py (parte 5/5)

```
    base64_strings      : ['~Z0', 'r+r', 'o+^', 'Hp5', 'dDN', '~)^', '~)e',
    ↪  '.+-', '%CD', "F'^", '5]v', '~)eO*^', '9P\x0b', 'D@\x0b', '\x0c"\x0c',
    ↪  'Z!b', '<A\n', '0CT', 'DCS', 'r+^', 'jw]', 'jwo', 'nz-', 'v72', "\r'2",
    ↪  'w72', 'x:-', 'zk)', 'z{)', ' G2', '(w2', '(x+', '8x+', '<x+', '=(+',
    ↪  'Hw2', 'Lx+', "M'2", 'aG2', 'dw2', '0au', 'HCC', 'Q1|', 'v+$', 'X@O',
    ↪  ',AS', '\t@@', '<DT', 'MDJ0CHI0\r']
    gps                 : True (Use of GPS noticed in assets, libraries or other
    ↪  unusual directories)
    mms                 : True (Probably dealing with MMS)
    play_services       : True (Uses Google Play services)
    qemu                : True (Detection of QEMU emulator)
            DEX properties /  About classes.dex format
    magic               : 35

                   Kit properties /  Detected 3rd party SDKs
    bouncycastle        : True (Bouncycastle crypto provider)
    firebase            : True
    googlegms           : True
    googlegson          : True
    jetbrains           : True
    material            : True (Material.IO design system)
    mixpanel            : True
    netty               : True (Netty - asynchronous event-driven app framework)
    okhttp              : True (Square OkHttp client for Android)
    okio                : True (Square I/O API for Java)
    protobuf            : True
    zxing               : True (Zebra Crossing 1D/2D barcode image processing)
    END
```

A.3.2 *Burpsuite*

Esta secção descreve em maior profundidade os resultados que foram obtidos no decorrer do uso da aplicação. Estes são utilizados durante a análise como forma de analisar o conteúdo da informação que está a ser trocado entre o DM e destino remoto.

Listagem 53: Dados recolhidos - POST (host: user.live.boltsvc.net) - Parte (1/3)


```
"json_data": {
"64": "192.168.1.72",
"1": "Xiaomi",
"2": "Redmi Note 3",
"3": "msm8952",
"4": "unknown",
"5": "Xiaomi",
"6": [
    "arm64-v8a",
    "armeabi-v7a",
    "armeabi"
],
"7": "kenzo",
"8": "Xiaomi/kenzo/kenzo:6.0.1/MMB29M/V8.2.1.0.MHOCNDL:user/release-keys",
"9": "qcom",
"10": "PQ3A.190801.002",
"11": "976_GEN_PACK-1.103654.1.105572.1,976_GEN_PACK-1.103654.1.105572.1",
"13": 28,
"14": "9",
"17": 20876701696,
"18": 26283376640,
"19": 20876701696,
"20": 26283376640,
"68": "Linux version 3.10.108-Escrima-X28-Kamui-LightningBlade-PurpleLightning
↪  (McFy@Android-A320FL-Build-Box) (gcc version 4.9.x 20150123 (prerelease)
↪  (GCC) ) #1 SMP PREEMPT Fri Sep 6 09:56:13 CEST 2019",
"16": 1.0,
"28": 0,
"21": 3.0,
"22": 480,
"23": 1920,
"24": 3.0,
"25": 1080,
"26": 391.885,
"27": 381.0,
"72": {
    "BMI160 Accelerometer": "BOSCH/2061000",
    "BMI160 Accelerometer Uncalibrated": "BOSCH/2061000",
    "YAS537 Magnetometer": "Yamaha/35193090",
    "YAS537 Magnetometer Uncalibrated": "Yamaha/35193090",
    "BMI160 Gyroscope": "BOSCH/2061000",
    "BMI160 Gyroscope Uncalibrated": "BOSCH/2061000",
    "LTR55X ALSPRX": "Liteon/1",
    "BMI160 Accelerometer -Wakeup Secondary": "BOSCH/2061000",
    "BMI160 Accelerometer Uncalibrated -Wakeup Secondary": "BOSCH/2061000",
    "YAS537 Magnetometer -Wakeup Secondary": "Yamaha/35193090",
    "YAS537 Magnetometer Uncalibrated -Wakeup Secondary": "Yamaha/35193090",

                        /*----Continua próxima página----*/
```

Listagem 54: Dados recolhidos - POST (host: user.live.boltsvc.net) - Parte (2/3)

```
        /*----Continuação-----*/
        "BMI160 Gyroscope -Wakeup Secondary": "BOSCH/2061000",
        "BMI160 Gyroscope Uncalibrated -Wakeup Secondary": "BOSCH/2061000",
        "LTR55X ALSPRX -Non Wakeup Secondary": "Liteon/2",
        "LTR55X ALSPRX -Wakeup Secondary": "Liteon/1",
        "Gravity": "QTI/2",
        "Linear Acceleration": "QTI/2",
        "Rotation Vector": "QTI/2",
        "Step Detector": "QTI/2",
        "Step Counter": "QTI/2",
        "Significant Motion Detector": "QTI/2",
        "Game Rotation Vector": "QTI/2",
        "GeoMagnetic Rotation Vector": "QTI/2",
        "Orientation": "QTI/2",
        "Tilt Detector": "QTI/2",
        "Gravity -Wakeup Secondary": "QTI/2",
        "Linear Acceleration -Wakeup Secondary": "QTI/2",
         "Rotation Vector -Wakeup Secondary": "QTI/2",
        "Step Detector -Wakeup Secondary": "QTI/2",
        "Step Counter -Wakeup Secondary": "QTI/2",
        "Game Rotation Vector -Wakeup Secondary": "QTI/2",
        "GeoMagnetic Rotation Vector -Wakeup Secondary": "QTI/2",
        "Orientation -Wakeup Secondary": "QTI/2",
        "AMD": "QTI/2",
        "RMD": "QTI/2",
        "Basic Gestures": "QTI/2",
        "Facing": "QTI/2",
        "Pedometer": "QTI/2",
        "Motion Accel": "QTI/2",
        "Coarse Motion Classifier": "QTI/3"
        },
        "73": {
                "level": 100,
                "scale": 100,
                "status": 5,
                "charging": true,
                "chargeplug": 2,
                "temperature": 315,
                "voltage": 4321,
                "health": 2
        },
        "36": "667c3d00950adad4",
        "33": "",
        "34": "",
        "87": "",
        "49": "e2:b9:e5:b5:c9:5b",
        "50": "\"MEO-B5C95B-5G\"",
        "51": "02:00:00:00:00:00",
        "52": 1208068288,
        "53": -54,
        "54": 433,
        "55": "192.168.1.72",
        "56": "192.168.1.254",
        "57": "192.168.1.254",
        "58": "0.0.0.0",
        "59": "192.168.1.254",
        "60": 0,
                        /*----Continua próxima página----*/
```

Listagem 55: Dados recolhidos - POST (host: user.live.boltsvc.net) - Parte (3/3)

```
        /*----Continuação-----*/
        "61": 3600,
        "62": "WIFI",
        "63": "",
        "65": "64:CC:2E:A8:61:B9",
        "66": "fe80::66cc:2eff:fea8:61b9",
        "67": "192.168.1.72",
        "37": 727173708,
        "38": 313481673,
        "39": 28,
        "40": 28409616,
        "41": 6904824,
        "42": 6,
        "43": "PT",
        "44": "pt",
        "45": "Europe/Lisbon",
        "46": "Linux",
        "47": "aarch64",
        "48": "3.10.108-Escrima-X28-Kamui-LightningBlade-PurpleLightning",
        "69": [
        "/system/framework/services.jar:/system/framework/ethernet-service.jar:/system/framework/wifi-service.jar:/system/framework/com.android.location.provider.jar",
        "/sbin:/system/sbin:/system/bin:/system/xbin:/odm/bin:/vendor/bin:/vendor/xbin",
        "11",
        "/cache",
        "/data",
        "/system/etc/terminfo",
        "/mnt/asec",
        "/sdcard",
        "1",
        "/system/app",
        "/storage",
        "/system",
        "/data/cache",
        "/system/framework/core-oj.jar:/system/framework/core-libart.jar:/system/framework/conscrypt.jar:/system/framework/okhttp.jar:/system/framework/bouncycastle.jar:/system/framework/apache-xml.jar:/system/framework/ext.jar:/system/framework/framework.jar:/system/framework/telephony-common.jar:/system/framework/voip-common.jar:/system/framework/ims-common.jar:/system/framework/android.hidl.base-V1.0-java.jar:/system/framework/android.hidl.manager-V1.0-java.jar:/system/framework/framework-oahl-backward-compatibility.jar:/system/framework/android.test.base.jar:/system/framework/QPerformance.jar:/system/framework/UxPerformance.jar:/system/framework/telephony-ext.jar"
        ],
        "70": 18835,
        "71": "ee.mtakso.client",
        "79": "fVSmHNoDS520w3wAbPaKWL",
        "80": true,
        "85": 39.63221,
        "86": -8.831129
```

B

APÊNDICE B

B.1 OBJECTION

Existe uma quantidade grande de informação que é possível obter através desta ferramenta, na qual se considera relevante para ser possível perceber a lógica da aplicação. É apenas com a informação que se encontra de seguida que pode ser possível explorar de forma dinâmica a aplicação.

B.1.1 *Lista de Activities - Bolt*

É possível obter a lista de todas as *activities* sobre as quais é possível realizar *hook* da aplicação em estudo.

Existem **25 classes** associadas a *Activities* sobre as quais é possível realizar *hook*:

- `co.paystack.android.ui.AddressVerificationActivity`
- `co.paystack.android.ui.AuthActivity`
- `co.paystack.android.ui.CardActivity`
- `co.paystack.android.ui.OtpActivity`
- `co.paystack.android.ui.PinActivity`
- `com.adyen.threeds2.internal.ui.activity.ChallengeActivity`
- `com.clevertap.android.sdk.InAppNotificationActivity`
- `com.clevertap.android.sdk.inbox.CTInboxActivity`
- `com.facebook.CustomTabActivity`
- `com.facebook.CustomTabMainActivity`
- `com.facebook.FacebookActivity`
- `com.facebook.notifications.internal.activity.CardActivity`

- com.google.android.gms.auth.api.signin.internal.SignInHubActivity
- com.google.android.gms.common.api.GoogleApiActivity
- com.google.android.play.core.common.PlayCoreDialogWrapperActivity
- com.patloew.rxlocation.LocationSettingsActivity
- com.survicate.surveys.SurveyActivity
- com.veriff.sdk.views.base.verification.VeriffActivity
- ee.mtakso.client.activity.SplashHomeActivity
- ee.mtakso.client.activity.ThreeDSActivity
- ee.mtakso.client.activity.voip.VoipCallActivity
- ee.mtakso.client.newbase.RideHailingMapActivity
- ee.mtakso.client.newbase.deeplink.AppsFlyerDeeplinkActivity
- ee.mtakso.client.newbase.deeplink.DeeplinkActivity
- eu.bolt.verification.sdk.VerificationSDKActivity

B.1.2 *Lista de Serviços*

Para a lista de serviços foram encontradas **20 classes**:

- androidx.room.MultiInstanceInvalidationService
- androidx.work.impl.background.systemjob.SystemJobService
- androidx.work.impl.foreground.SystemForegroundService
- com.clevertap.android.sdk.pushnotification.amp.CTBackgroundIntentService
- com.clevertap.android.sdk.pushnotification.amp.CTBackgroundJobService
- com.google.android.datatransport.runtime.backends.TransportBackendDiscovery
- com.google.android.datatransport.runtime.scheduling.jobscheduling.JobInfoSchedulerService

- com.google.android.gms.auth.api.signin.RevocationBoundService
- com.google.android.gms.common.internal.zzo
- com.google.android.gms.measurement.AppMeasurementJobService
- com.google.android.gms.measurement.AppMeasurementService
- com.google.firebase.components.ComponentDiscoveryService
- com.google.firebase.messaging.FirebaseMessagingService
- com.google.mlkit.common.internal.MlKitComponentDiscoveryService
- com.sinch.android.rtc.internal.client.fcm.InstanceIDTokenService
- com.veriff.sdk.service.SendAuthenticationFlowDataToServerService
- ee.mtakso.client.fcm.FcmListenerService
- ee.mtakso.client.ribs.root.ridehailing.safetytoolkit.sosintegration.service.IncidentReportingService
- eu.bolt.voip.service.VoipService
- mr

B.1.3 *Lista de Receivers*

Foram encontradas 14 classes que estão associadas a *receivers*:

- androidx.work.impl.diagnostics.DiagnosticsReceiver
- androidx.work.impl.utils.ForceStopRunnable\$BroadcastReceiver
- ap
- com.adyen.threeds2.internal.AppUpgradeBroadcastReceiver
- com.appsflyer.MultipleInstallBroadcastReceiver
- com.clevertap.android.sdk.pushnotification.CTPushNotificationReceiver
- com.facebook.CurrentAccessTokenExpirationBroadcastReceiver
- com.google.android.datatransport.runtime.scheduling.jobscheduling.AlarmManagerSchedulerBroadcastReceiver

- `com.google.android.gms.measurement.AppMeasurementReceiver`
- `com.google.firebase.iid.FirebaseInstanceIdReceiver`
- `du`
- `eu.bolt.client.chat.notifications.NotificationDeleteIntentReceiver`
- `y7.c`
- `zo`

B.2 SCRIPTS - FRIDA

O *script* ilustrado na Listagem 56 pode ser utilizado para obter informação detalhada sobre uma chave presente na *KeyStore*, sendo necessário conhecer o nome do alias que é indicado como argumento no cabeçalho desta função.

O *script* original pode ser obtido a partir de https://github.com/WithSecureLabs/android-keystore-audit/blob/master/frida-scripts/tracer-cipher.js, fica o código mais relevante que foi utilizado com as modificações que foram efetuadas presentes na Listagem 57 e 58.

O *script* na listagem 59 é um exemplo adaptado que foi utilizado durante os testes https://infosecwriteups.com/analyzing-android-encryption-processes-with-frida-a3ab2622fce9 a este caso de uso.

Listagem 56: Obter mais informação sobre a chave com o alias - Authenticator

```
function AliasInfo('Authenticator') {
var result = {};
Java.perform(function () {
        var keyStoreCls = Java.use('java.security.KeyStore');
        var keyFactoryCls = Java.use('java.security.KeyFactory');
        var keyInfoCls = Java.use('android.security.keystore.KeyInfo');
        var keySecretKeyFactoryCls = Java.use('javax.crypto.SecretKeyFactory');
        var keyFactoryObj = null;
        var keyStoreObj = keyStoreCls.getInstance('AndroidKeyStore');
        keyStoreObj.load(null);
        var key = keyStoreObj.getKey('Authenticator', null);
        if (key == null) {
                console.log('key does not exist');
                return null; }
        try {
                keyFactoryObj = keyFactoryCls.getInstance(key.getAlgorithm(),
                ↪  'AndroidKeyStore');
        } catch (err) {
                keyFactoryObj =
                ↪  keySecretKeyFactoryCls.getInstance(key.getAlgorithm(),
                ↪  'AndroidKeyStore'); }
        var keyInfo = keyFactoryObj.getKeySpec(key, keyInfoCls.class);
        result.keyAlgorithm = key.getAlgorithm();
        result.keySize = keyInfoCls['getKeySize'].call(keyInfo);
        result.blockModes = keyInfoCls['getBlockModes'].call(keyInfo);
        result.digests = keyInfoCls['getDigests'].call(keyInfo);
        result.encryptionPaddings =
        ↪  keyInfoCls['getEncryptionPaddings'].call(keyInfo);
        result.keyValidityForConsumptionEnd =
        ↪  keyInfoCls['getKeyValidityForConsumptionEnd'].call(keyInfo);
        if (result.keyValidityForConsumptionEnd != null)
        ↪  result.keyValidityForConsumptionEnd =
        ↪  result.keyValidityForConsumptionEnd.toString();
        result.keyValidityForOriginationEnd =
        ↪  keyInfoCls['getKeyValidityForOriginationEnd'].call(keyInfo);
        if (result.keyValidityForOriginationEnd != null)
        ↪  result.keyValidityForOriginationEnd =
        ↪  result.keyValidityForOriginationEnd.toString();
        result.keyValidityStart =
        ↪  keyInfoCls['getKeyValidityStart'].call(keyInfo);
        if (result.keyValidityStart != null) result.keyValidityStart =
        ↪  result.keyValidityStart.toString();
        re2sult.keystoreAlias = keyInfoCls['getKeystoreAlias'].call(keyInfo);
        result.origin = keyInfoCls['getOrigin'].call(keyInfo);
        result.purposes = keyInfoCls['getPurposes'].call(keyInfo);
        result.signaturePaddings =
        ↪  keyInfoCls['getSignaturePaddings'].call(keyInfo);
        result.userAuthenticationValidityDurationSeconds = keyInfoCls['getUserAu
        ↪  thenticationValidityDurationSeconds'].call(keyInfo);
        result.isInsideSecureHardware =
        ↪  keyInfoCls['isInsideSecureHardware'].call(keyInfo);
        result.isInvalidatedByBiometricEnrollment =
        ↪  keyInfoCls['isInvalidatedByBiometricEnrollment'].call(keyInfo);
        try { result.isTrustedUserPresenceRequired =
        ↪  keyInfoCls['isTrustedUserPresenceRequired'].call(keyInfo); } catch
        ↪  (err) { }
        result.isUserAuthenticationRequired =
        ↪  keyInfoCls['isUserAuthenticationRequired'].call(keyInfo);
        result.isUserAuthenticationRequirementEnforcedBySecureHardware =
        ↪  keyInfoCls['isUserAuthenticationRequirementEnforcedBySecureHardware'
        ↪  ].call(keyInfo);
        result.isUserAuthenticationValidWhileOnBody =
        ↪  keyInfoCls['isUserAuthenticationValidWhileOnBody'].call(keyInfo);
        try { result.isUserConfirmationRequired =
        ↪  keyInfoCls['isUserConfirmationRequired'].call(keyInfo); } catch
        ↪  (err) { }
});
return result;
}
```

Listagem 57: Manipular métodos de encriptação - Google Authenticator (parte 1/2)

```
Java.perform(function () {
        hookCipherGetInstance();
        hookCipherInit4();
        hookDoFinal5();
});
function hookCipherGetInstance() {
var cipherGetInstance =
↪  Java.use('javax.crypto.Cipher')['getInstance'].overload("java.lang.String");
cipherGetInstance.implementation = function (type) {
        console.log("[Cipher.getInstance()]: type: " + type);
        var tmp = this.getInstance(type);
        console.log("[Cipher.getInstance()]:  cipherObj: " + tmp);
        cipherList.push(tmp);
        return tmp;
}
}
function hookCipherInit4() {
var cipherInit = Java.use('javax.crypto.Cipher')['init'].overload('int',
↪  'java.security.Key', 'java.security.spec.AlgorithmParameterSpec');
cipherInit.implementation = function (mode, secretKey, spec) {
        console.log("[Cipher.init4()]: mode: " + decodeMode(mode) + ", secretKey:
        ↪  " + secretKey.$className + " spec:" + spec + " , cipherObj: " +
        ↪  this);
        console.log("\n ------Aqui------  \n");
        console.log("Algoritmo: " + secretKey.getAlgorithm());
        console.log("Key Encoded :"+ secretKey.getEncoded());
        console.log("Key toString :" + secretKey.toString());
        console.log("\n ------Termina------  \n");
        var tmp = this.init(mode, secretKey, spec);        }
}
function hookDoFinal5() {
var cipherInit = Java.use('javax.crypto.Cipher')['doFinal'].overload('[B',
↪  'int', 'int');
cipherInit.implementation = function (byteArr, a1, a2) {
        console.log("[Cipher.doFinal5()]: " + "  cipherObj: " + this);
        dumpByteArray('In buffer (cipher: ' + this.getAlgorithm() +')', byteArr);
        console.log("Comecar a partir da posicao do Offset : "+ a1);
        console.log("Terminar na posicao do Offset : "+ a2);
        var tmp = this.doFinal(byteArr, a1, a2);
        dumpByteArray('Out buffer (cipher: ' + this.getAlgorithm() +')', tmp);
        return tmp; }
}
function decodeMode(mode) {
        if (mode == 1)
        return "Encrypt mode";
        else if (mode == 2)
        return "Decrypt mode";
        else if (mode == 3)
        return "Wrap mode";
        else if (mode == 4)
        return "Unwrap mode";
}
/* All below is hexdump implementation*/
function dumpByteArray(title, byteArr) {
if (byteArr != null) {
        try {
                var buff = new ArrayBuffer(byteArr.length)
                var dtv = new DataView(buff)
                for (var i = 0; i < byteArr.length; i++) {
                        dtv.setUint8(i, byteArr[i]); // Frida sucks sometimes and
                        ↪  returns different byteArr.length between
                        ↪  ArrayBuffer(byteArr.length) and for(..; i <
                        ↪  byteArr.length;..). It occured even when Array.copyOf
                        ↪  was done to work on copy.
                }
                console.log(title + ":\n");
                console.log(hexdumpJS(dtv.buffer, 0, byteArr.length))
        } catch (error) { console.log("Exception has occured in hexdump")
        ↪  }        }
else {console.log("byteArr is null!");}
}
```

Listagem 58: Manipular métodos de encriptação - Google Authenticator (parte 2/2)

```
function _fillUp(value, count, fillWith) {
var l = count - value.length;
var ret = "";
while (--l > -1)
ret += fillWith;
return ret + value;
}
function hexdumpJS(arrayBuffer, offset, length) {

var view = new DataView(arrayBuffer);
offset = offset || 0;
length = length || arrayBuffer.byteLength;

var out = _fillUp("Offset", 8, " ") + "  00 01 02 03 04 05 06 07 08 09 0A 0B 0C
↪  0D 0E 0F\n";
var row = "";
for (var i = 0; i < length; i += 16) {
        row += _fillUp(offset.toString(16).toUpperCase(), 8, "0") + "  ";
        var n = Math.min(16, length - offset);
        var string = "";
        for (var j = 0; j < 16; ++j) {
                if (j < n) {
                        var value = view.getUint8(offset);
                        string += (value >= 32 && value < 128) ?
                        ↪  String.fromCharCode(value) : ".";
                        row += _fillUp(value.toString(16).toUpperCase(), 2, "0")
                        ↪  + " ";
                        offset++;
                }
                else {
                        row += "   ";
                        string += " ";
                }
        }
        row += " " + string + "\n";
}
out += row;
return out;
};
```

Listagem 59: Aceder à chave, vetor de inicialização e segredo

```
Java.perform(function(){
        var base64 = Java.use('java.util.Base64');
        var cipher = Java.use("javax.crypto.Cipher");

        cipher.init.overload('int', 'java.security.Key',
        ↪  'java.security.spec.AlgorithmParameterSpec').implementation =
        ↪  function(opmode,key,algorithmParameter){
                send_log("Chave",base64.getEncoder().encodeToString(key.getEncod
                ↪  ed()));
                send_log("Modo de Operacao",this.getOpmodeString(opmode));
                send_log("Algoritmo",this.getAlgorithm())
                this.init.overload('int', 'java.security.Key',
                ↪  'java.security.spec.AlgorithmParameterSpec').call(this,opmod
                ↪  e,key,algorithmParameter)
        }
        var ivParameter =  Java.use('javax.crypto.spec.IvParameterSpec');
        ivParameter.$init.overload('[B').implementation = function(ivKey){
                send_log("Vetor
                ↪  Inicializacao",base64.getEncoder().encodeToString(ivKey))
                this.$init.overload('[B').call(this,ivKey)
        }

        cipher.doFinal.overload("[B").implementation = function(input){
                var input_base64 = base64.getEncoder().encodeToString(input)
                var input_string = byte_to_string(input)
                var output = this.doFinal.overload('[B').call(this,input)
                var output_base64 = base64.getEncoder().encodeToString(output)
                send_log("Input Base64",input_base64)
                send_log("Input String",input_string)
                send_log("Output Base64",output_base64)
                return output;
        }
})
function send_log(string, value){
        console.log("[+] "+string+" : "+value);
}
function byte_to_string(byte_array){
        var StringClass = Java.use('java.lang.String');
        return StringClass.$new(byte_array).toString();
}
```

C

APÊNDICE C

C.1 CASO DE USO: GOOGLE AUTHENTICATOR

Este apêndice serve para familiarizar o leitor com as capacidades do `Frida`, através de uma aplicação menos complexa. Apenas para situar o leitor é apresentada de seguida uma explicação no caso de não conhecer o funcionamento da aplicação *Google Authenticator*.

A aplicação *Google Authenticator* gera códigos (One-Time Password (OTP)) para serem utilizados como segundo fator de autenticação. Os códigos podem também chamados de *tokens* de autenticação, são apresentados em conjunto com o nome das contas associadas. O código é composto por um número inteiro de 6 dígitos e este é regenerado a cada 30 segundos. Os códigos (OTP) são gerados de forma independente para cada uma das contas adicionadas. Só podem ser adicionadas contas (serviços) que utilizem Two-Factor Authentication (2FA) e os quais utilizem algoritmos Time-based One-time Password (TOTP) ou HMAC-Based One-time Password (HOTP), para que sejam compatíveis com os utilizados pelo *Google Authenticator*.

A figura 45 permite-nos observar o processo que é realizado para gerar e guardar as chaves secretas. Tal como podemos observar pela figura, existem duas formas de obter a chave secreta, a primeira forma apresentada é obtida através da leitura de um *QR Code*, a segunda através da introdução manual da chave gerada na aplicação. Nesta segunda forma podemos indicar se o código a gerar é com base na hora (TOTP) ou num contador (HOTP).

A figura 46, descreve o processo que é utilizado para validar os códigos gerados. Se o processo for bem sucedido a autenticação do utilizador é validada. Em alguns casos a autenticação pode falhar devido ao curto espaço de tempo de validade dos códigos gerados. Por vezes os servidores aceitam códigos que tenham sido gerados na iteração anterior devido ao possível atraso que haja na validação do processo que observámos na figura 46.

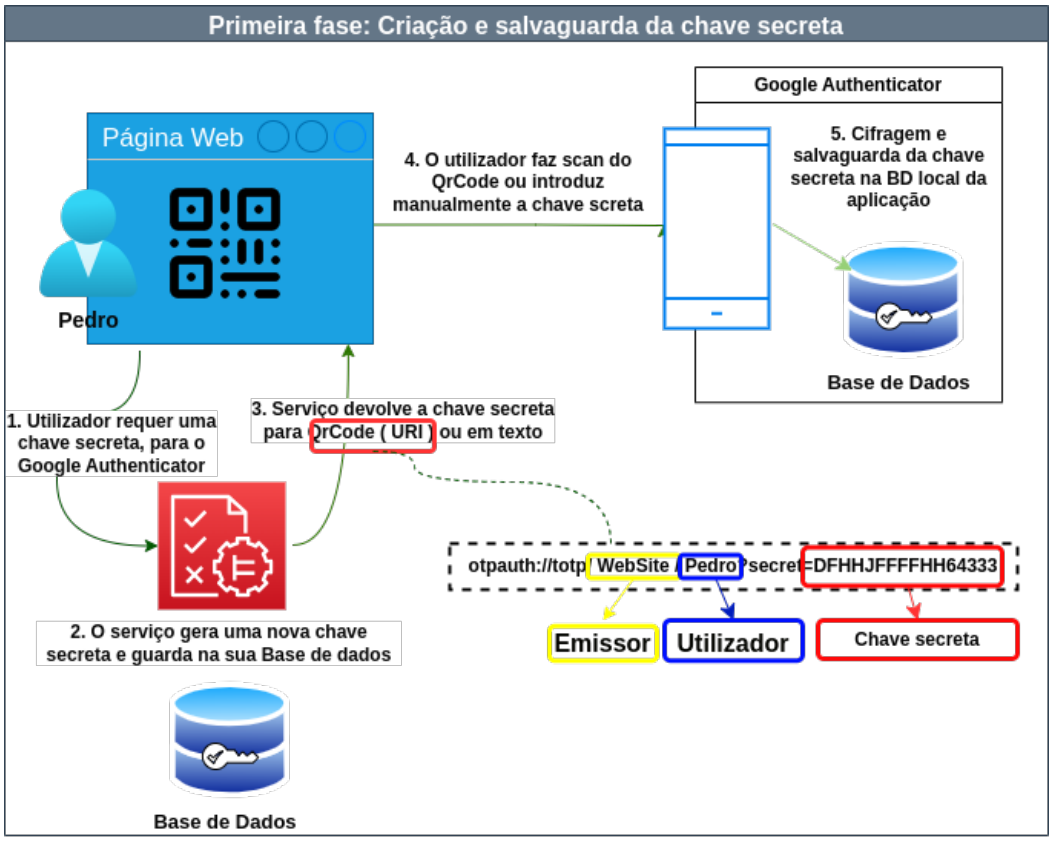


Figura 45: Obter chave secreta

Um dos objetivos deste caso de uso é perceber se conseguimos obter o "segredo", utilizado para gerar os códigos de autenticação. Outro objetivo é descobrir de que forma este segredo é guardado localmente e se é possível reverter o processo recorrendo ao `Frida`.

Para isso começamos por utilizar o `Frida` para explorar o código fonte da aplicação (sem conhecer o código existente). Para isso começamos por listar classes e métodos existentes 60, apesar de não ser a forma mais fácil de analisar a mesma (em aplicações complexas), é possível de uma forma mais rápida ter acesso à nomenclatura (classes e métodos) utilizada pela aplicação.

Com esta breve análise podemos verificar se existe ou não ofuscação, no caso de não existir (ofuscação), perceber quais os métodos que nos despertem mais a atenção para serem alvos de uma análise mais profunda.

Ao listar as classes observamos que o *package*:`com.google.android.apps.authenticator.otp`, contém as diversas classes responsáveis pelas funções de manipulação dos códigos OTP. Após conhecer ter um ideia das possíveis classes que estamos interessados partimos para a segunda parte que é conhecer os métodos que fazem

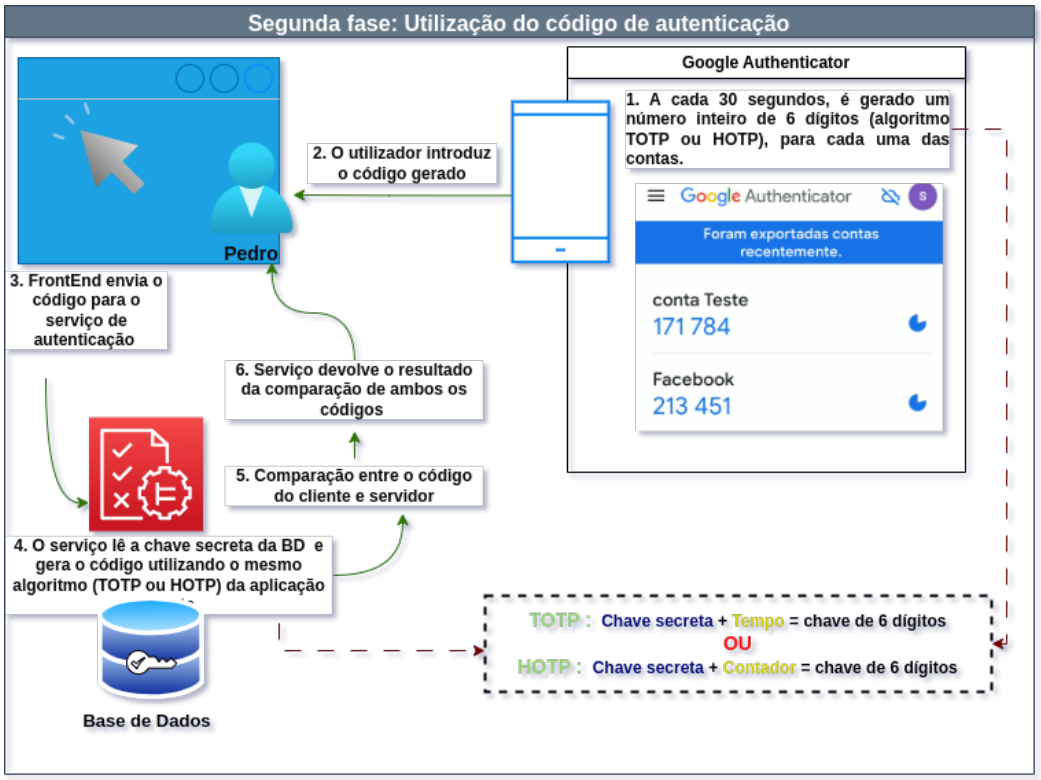


Figura 46: Processo de validação dos códigos gerados

parte das mesmas. Aqui mais uma vez é importante ter uma visão genérica do que está a ser feito em cada uma das classes, até encontrar os métodos que estamos interessados.

Após utilizar a segunda parte do exemplo apresentado na Listagem 60 para observar as classes pretendidas (`PinInfo` e `AccountDb$AccountIndex`) e obter as respetivas listas de métodos e seus parâmetros é possível começar a instrumentalização da aplicação.

Métodos para gestão de contas - Google Authenticator

- Package: com.google.android.apps.authenticator.otp;
- Classe: AccountDb;
  - Método: `getName(AccountDb$AccountIndex)`
  - Método: `getSecret(AccountDb$AccountIndex)`
- Classe: PinInfo;
  - Método: getPin()
  - Método: getIndex()

Listagem 60: Script: enumerar classes e métodos - Google Authenticator

```
/**
* Comando -> frida -U -f com.google.android.apps.authenticator2 -l
↪  listarClasses.js -o classesGoogleAuthenticator
\texttt{}*
* Enumerar classes carregadas - genérico:
*/
Java.perform(function () {
        Java.enumerateLoadedClasses({
                onMatch: function (c) {
                        console.log(c);
                },
                onComplete: function(d){
                }
        });
});

/*
* Guardar métodos da classe em analise:
* frida -U -F -l listarMetodosDaClasse.js -o metodosClassePinInfo
*
* Obter lista de metodos declarados - Google Authenticator
*/
Java.perform(function () {
        const hook =
        ↪  Java.use('com.google.android.apps.authenticator.otp.PinInfo');

        var methods = hook.class.getDeclaredMethods();

        methods.forEach(i => {
                console.log("Metodo : " + i + "\n");
        });
        hook.$dispose();
});
```

Após a sua identificação procedemos com a leitura dos valores devolvidos pelos métodos anteriormente referidos. A melhor forma de o fazer é aceder às instâncias em memória, respeitantes às classes já referidas. Para melhor contextualizar o leitor, o *Frida* consegue manipular a execução da aplicação de diferentes formas. Podemos se assim for desejado alterar o funcionamento do método pretendido. O que na realidade ao invés de reescrever o código existente, o *Frida* injeta o novo código pretendido. Todo este processo é completamente transparente ao utilizador, o que permite executar alterações sem existir necessidade de compilação da aplicação e/ou remoção do código alterado.

Por forma a perceber o que foi feito vamos resumir o *script*, apresentado na Listagem 61. Começámos por identificar o caminho completo (*package name* + nome da classe) com o qual pretendemos interagir. A sintaxe (`Java.choose(className, callbacks)`) permite enumerar todas instâncias 'correntes' na `Java heap` que correspondam ao nome da classe. O `callbacks` é o objeto da classe de cada instância da classe que será criada ao longo da execução da aplicação.

Listagem 61: Script Frida - obter chaves secretas e Pin Codes - Google Authenticator

```
Java.perform(function () {
        /**
        * Classe a procurar na pilha Java em memória
        */
        const accountDbClass =
        ↪  'com.google.android.apps.authenticator.otp.AccountDb$AccountIndex';

        Java.choose(accountDbClass, {
                /**
                * @param {Chamada por cada instancia em uso encontrada} account
                */
                onMatch: function (account) {
                        console.log("Nome da Conta: " + account.name.value);
                        console.log("Segredo : "+ account.secret.value + "\\n");
                },
                /**
                * onComplete - Chamado após todas as intancias terem sido
                ↪  enumeradas
                */
                onComplete: function (ret) {
                }
        })

        const pinInfoClass = 'com.google.android.apps.authenticator.otp.PinInfo'

        Java.choose(pinInfoClass, {
                onMatch: function (instance) {
                        //Obter o valor de retorno das funções pinInfoClass e
                        ↪  getPin
                        console.log("Nome da Conta : "+ instance.getIndex());
                        console.log("PinCode : " + instance.getPin() + "\\n");
                },
                onComplete: function (ret) {
                }
        })
})
```

O `onMatch` é chamado cada vez que uma dessas instâncias é encontrada e que podem ser utilizadas. É neste método que colocámos o código que pretendemos executar, tendo a noção que este será executado cada vez que existir a nova chamada do mesmo. Tal permite o acesso a diferentes estados durante a execução da aplicação, sejam estes métodos, variáveis, condições, etc.. O `onComplete` é chamado após todas as instâncias terem sido enumeradas, e por norma é utilizado quando pretendemos executar alguma ação após o termino da manipulação da classe. O seu uso não é obrigatório, ao contrário do `onMatch`, como tal pode ser omitido.

Para que seja facilitado o processo de análise dinâmica é importante consultar o código que foi obtido durante a análise estática. Nesta fase, mesmo que o código esteja ofuscado é importante conhecer a nomenclatura que é utilizada. Isto porque é necessário usar o mesmo nome, tal como está no código fonte para que tenhamos sucesso na instrumentalização da aplicação. No caso de uso encontrado em primeiro lugar procedemos à leitura dos valores das variáveis passados ao construtor da classe interna `AccountIndex`.

Para ler os valores das variáveis locais, faz-se uso da sintaxe `account.name.value`. Para ser mais fácil perceber a sintaxe vamos apresentar da seguinte forma:

- **account:** Instância do objeto (`hook`);
- **name:** Nome da variável local (igual ao código fonte);
- **value:** *Keyword* utilizada para aceder ao valor de uma variável;

A segunda parte da listagem 61 contém o código necessário para obter o *pinCode* que é nada mais que o OTP a ser usado como 2FA. A diferença que aqui encontramos é que ao invés de aceder ao valor de uma variável, vamos obter o valor devolvido pelos métodos pretendidos.

Começamos por identificar a classe que pretendemos fazer o `hook`, tal como foi feito anteriormente de forma a ter acesso ao objeto instanciado. Para obtermos o valor de retorno do "método", utilizamos a mesma sintaxe utilizada ao programar em *Java*. Nome do objeto instanciado, seguido de um ponto (.) e por fim o nome do método (`objeto.metodo()`).

Apresentamos os resultados tal como foram obtidos através da figura 47. Sendo estes apenas dados de contas para teste, não existe a necessidade de ocultar os mesmos.

Podemos encontrar um resumo dos resultados obtidos na listagem 62. Aqui temos os valores em texto legível da chave inicialmente utilizada para criar todos os OTP, dos quais são derivados e gerados pela aplicação. Este é portanto um dos pontos que exige uma maior segurança por parte da aplicação, e como podemos ver não é complexo a forma de se obter informação que é considerada crítica para o funcionamento e segurança desta aplicação.

Ao executar de novo o *script* verificamos que é apresentado o segredo codificado em base64, tal como vemos na mesma Listagem 62, ou na figura 47. Isto revela que a aplicação efectuou alterações ao valor guardado em memória do segredo. Aqui podemos ver O que é normal visto que se o atacante tiver permissões de leitura às BD locais e esta conter informação não cifrada, poderá por em causa a segurança das contas afetadas.

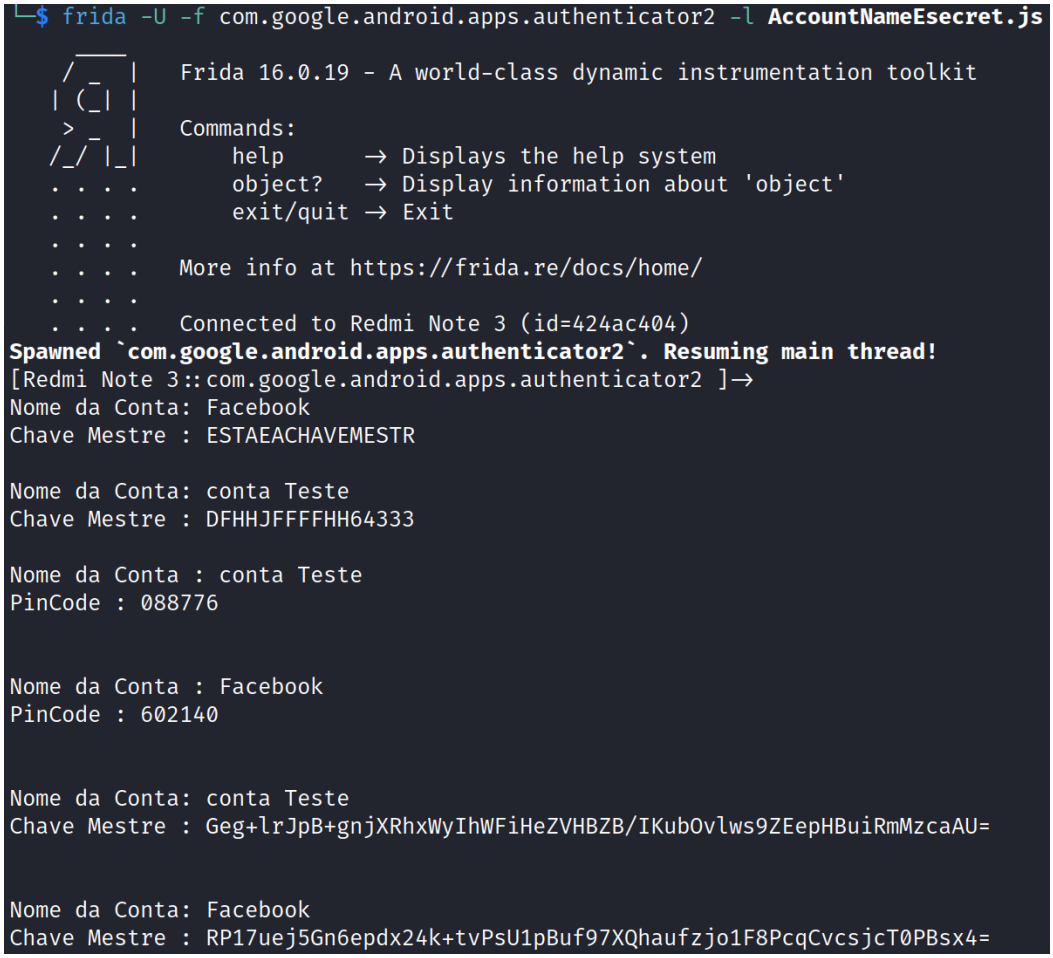


Figura 47: Execução do script e resultados obtidos

Listagem 62: Output Frida - Chave secreta e Pin Code - Google Authenticator

```
------------1º Execução--------------

Nome da Conta : Facebook
Segredo       : ESTAEACHAVEMESTR
PinCode       : 699261

Nome da Conta : conta Teste
Segredo       : DFHHJFFFFHH64333
PinCode       : 135438

------------2º Execução--------------

Nome da Conta   : Facebook
Segredo cifrado : RP17uej5Gn6epdx24k+tvPsU1pBuf97XQhaufzjo1F8PcqCvcsjcT0PBsx4=

Nome da Conta   : conta Teste
Segredo cifrado : Geg+lrJpB+gnjXRhxWyIhWFiHeZVHBZB/IKubOvlws9ZEepHBuiRmMzcaAU=
```

Por forma a confirmar o que já nos foi demonstrado ao executar o *script*, e com a ajuda da ferramenta `Objection`, podemos ler rapidamente os valores armazenados na tabela *accounts*. O procedimento já foi descrito em grande parte durante a explanação da ferramenta `Objection`, fica aqui um resumo dos pontos mais relevantes.

Listagem 63: Consultar bases de dados - Google Authenticator

```
...le.android.apps.authenticator2 on (Xiaomi: 9) [usb] # ls
Type  Last Modified           Read  Write  Hidden  Size       Name
----  ----------------------  ----  -----  ------  ---------  -------------
File  2023-06-14 13:05:16 GMT  True  True   False   36.0 KiB   databases
File  2023-06-20 21:03:16 GMT  True  True   False   402.4 KiB  databases-wal
File  2023-06-20 22:25:25 GMT  True  True   False   32.0 KiB   databases-shm

Readable: True  Writable: True
...le.android.apps.authenticator2 on (Xiaomi: 9) [usb] # sqlite connect databases
```

Após termos uma cópia das bases de dados locais podemos, executar a *query* à BD pretendida[1]. Podemos ver o conteúdo da tabela `"accounts"` da BD na figura 48.

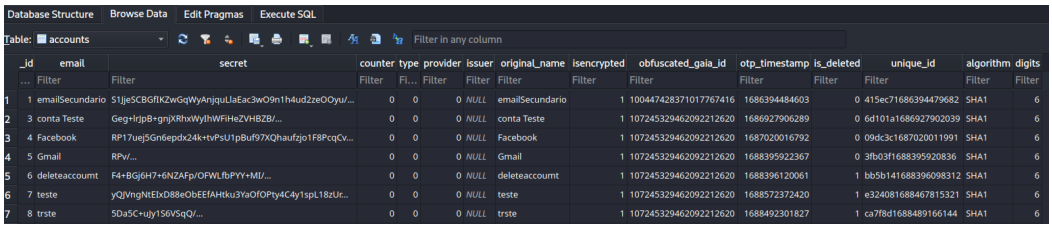


Figura 48: Tabela: accounts - Ferramenta: DB Browser SQLite - App: Google Authenticator

Os campos mais relevantes da tabela 'Accounts' são:

- **secret** : segredo cifrado;
- `original_name` : nome da conta;
- `otp_timestamp` : data e hora em Unix Epoch;
- `is_encrypted` : segredo está cifrado ou não;
- `is_deleted` : conta encontra-se ativa ou não;

Podemos encontrar as configurações desta BD através do separador "Edit Pragmas"ao utilizar a aplicação `DB Browser`, figura 49. Destas configurações, as mais relevantes no contexto de segurança é a validade da informação e a sua volatilidade. Na perspectiva forense, interessa saber como é feita a gestão dos registos da tabela. Ou seja se existe ou não a possibilidade de serem recuperados seja de forma parcial ou na sua totalidade. Para isso temos que verificar os campos `"Auto Vaccum"` e `"Secure Delete"`.

1 Adb-Extractor: https://github.com/labcif/ADB-Extractor

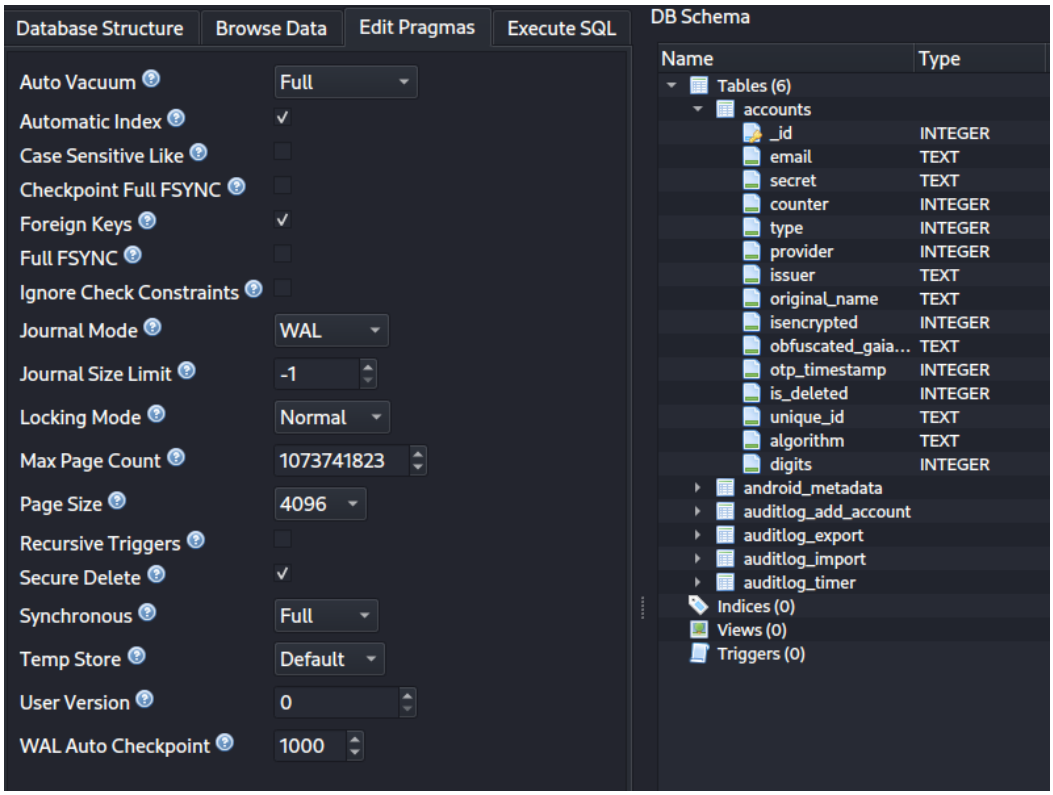


Figura 49: EditPragmas: Configurações da base dados local

O campo `"Auto Vaccum"` determina a gestão do espaço em disco da BD, o qual pode ser configurado com três opções distintas (*none, full* ou incremental). A BD está configurada com o valor `Full`, como podemos observar na figura 49. Esta configuração permite libertar espaço de forma automática cada vez que dados são removidos da mesma, no entanto esta opção não desfragmenta/reorganiza, nem compacta as páginas resultantes da operação[2].

O campo *"Secure Delete"*, também este define o comportamento ao remover conteúdo da base dados. Esta opção encontra-se ativa nesta BD, garantindo assim que os registos eliminados são reescritos por zero. Esta operação é efetuada cada vez que são realizadas operações de remoção ou atualização. E como consequência não deixa rasto dos valores alterados/removidos, o que dificulta ou nega a possibilidade de um analista forense obter o conteúdo dos valores apagados.

Ao observar os dados da tabela *Accounts*, no campo `secret` conseguimos responder à nossa dúvida anteriormente levantada, sobre a forma como segredo é guardado localmente. Podemos verificar que o mesmo se encontra cifrado e como tal não é possível reverter o processo facilmente.

2 Auto-Vacuum: https://www.sqlite.org/pragma.html#pragma_auto_vacuum - Acedido a 7 de Julho de 2023

Para podermos explorar como está a ser utilizada criptografia para converter o segredo inicial, precisamos em primeiro de conhecer o algoritmo e a chave que é utilizada no processo de cifragem.

Para nos auxiliar nessa tarefa podemos efetuar essa análise de duas formas, através da leitura e análise do código fonte da aplicação. Ou podemos tentar proceder ao `hook` das classes dos pacotes de criptografia do SO Android mais utilizados.

Para este caso de uso vamos utilizar ambos, no entanto optámos por começar por analisar o código fonte da aplicação *Google Authenticator*. Devido à pouca ofuscação existente é mais fácil navegar o mesmo para perceber o que é feito nas classes utilizadas para as tarefas de criptografia.

A classe '`AccountDb`' responsável por muita da manipulação dos valores de segredo e códigos de validação que vimos antes. No entanto para as tarefas de encriptação é utilizada a classe '`encryptUtil`'. A classe '`encryptUtil`' por sua vez contém informação sobre o processo de encriptação do segredo, e esta contém os métodos que realizam as tarefas de encriptação e desencriptação, desde logo podemos encontrar nas constantes declaradas com informação sobre as chaves a serem utilizadas.

- ACCOUNT_NAME = "Authenticator";
- ANDROID_KEY_STORE = "AndroidKeyStore";
- RSA_ALGORITHM = "RSA";

PROCESSO DE ENCRIPTAÇÃO DO SEGREDO Para simplificar a explicação do processo utilizado pela aplicação para a conversão do segredo no valor cifrado que encontrámos na BD `databases` da tabela `accounts`, foi criado o diagrama que é possível ver na figura 50.

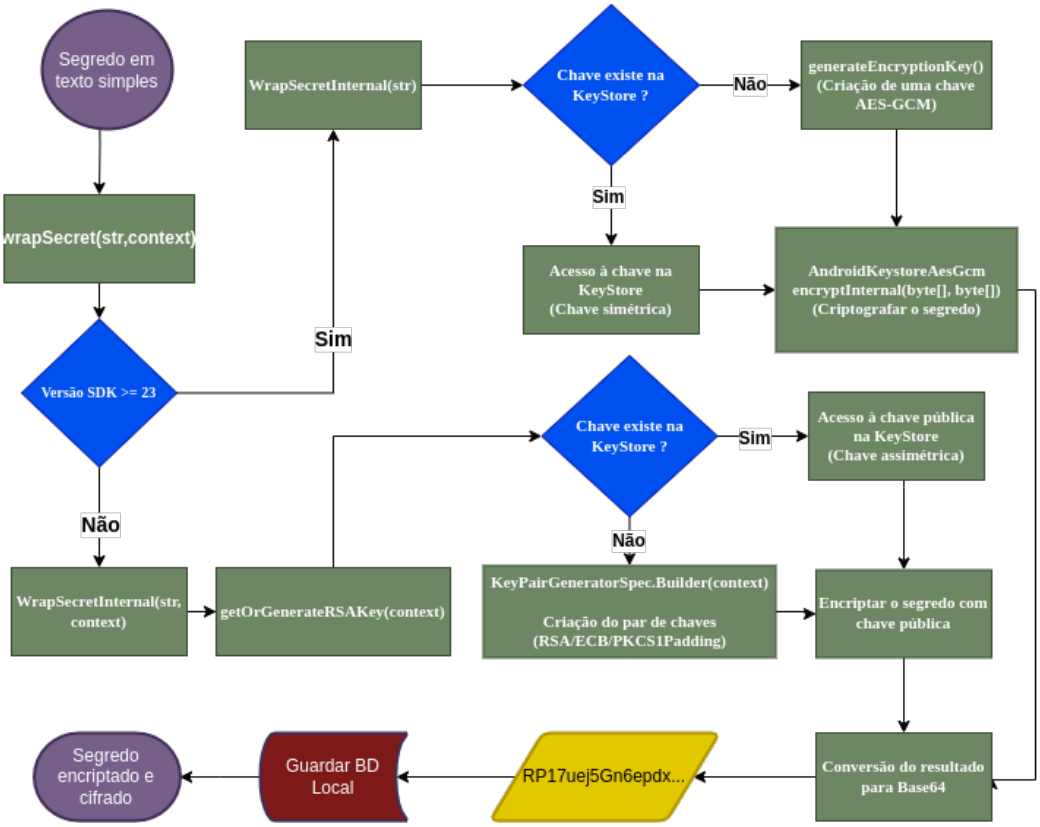


Figura 50: Diagrama do processo de encriptação do segredo

Como podemos ver através da figura 50, as grandes diferenças ocorrem consoante a versão do SDK. Esta figura contém apenas a descrição do processo de encriptação, para realizar o processo de obtenção do segredo inicial é possível de ser obtido com a reversão do processo aqui descrito. De realçar que ambas utilizam chaves que estão presentes na *Android KeyStore*, no entanto a grande diferença é que para versões anteriores ao SDK 28 é utilizada criptografia de chave assimétrica. Isto significa que é criado um par de chaves e utilizada a chave publica para encriptar o segredo. O que resulta na necessidade de ser utilizada a chave privada para reverter o texto cifrado para o seu valor original.

O dispositivo em uso durante a fase de teste da versão SDK 28 (Android 9.0), a aplicação utiliza encriptação simétrica para encriptar o segredo. Como tal é usada a mesma chave para encriptar e desencriptar o segredo, com a diferença que é utilizado um vetor de inicialização para gerar o texto cifrado. Se não obtivéssemos

esta informação no código fonte podemos também inquirir essa mesma informação diretamente à *KeyStore*. Neste caso teríamos primeiro que listar todas as entradas existentes, para isso utilizamos o *script* na Listagem 64

Listagem 64: Obter lista de Alias em uso - Google Authenticator

```
function hookKeystoreLoadStream(dump) {
        var keyStoreLoadStream = Java.use('java.security.KeyStore')['load'].over
        ↪  load('java.io.InputStream',
        ↪  '[C');

        keyStoreLoadStream.implementation = function (stream, charArray) {
                console.log("[Keystore.load(InputStream, char[])]: keystoreType:
                ↪  " + this.getType() + ", password: '" +
                ↪  charArrayToString(charArray) + "', inputSteam: " + stream);
                this.load(stream, charArray);
                if (dump) console.log(" Keystore loaded aliases: " +
                ↪  ListAliasesObj(this));
        }
}
```

O *script* apresentado na Listagem 64 permite-nos obter os nomes (*alias*) que estão a fazer uso das chaves presentes no sistema *Android Keystore*. Os *alias* são identificadores únicos que permitem evitar conflitos ou ambiguidades decorrentes do seu uso. Além disso permitem que não seja necessário conhecer os detalhes da sua implementação ou localização física da chave. Quando as chaves estão na *Keystore*, elas podem ser utilizadas para operações criptográficas, e o seu conteúdo não pode ser exportável[3]. Isto garante que na eventualidade de um processo ser comprometido por um atacante que faz uso de chaves presentes na *Keystore*, estas não poderão ser utilizadas fora do dispositivo *Android*.

Podemos inquirir mais informação sobre a chave que está a ser utilizada pela aplicação deste caso de uso, para isso mais uma vez utilizamos o *script* apresentado na Listagem 56. Neste *script*, apresentado pela Listagem 56, um dos campos mais relevantes nesta parte da análise é relativa ao método `"isInsideSecureHardware()"`. Este método indica-nos se a chave está vinculada ao hardware seguro, e dessa forma nunca pode ser acedida fora do hardware protegido. Acesso ao conteúdo (texto simples) da chave pode apenas ser feito por *threads* que estejam a ser executadas dentro do *secure hardware* e não pode ser exposto fora deste[4].

Podemos ver de seguida alguma da informação recolhida, observando-se que é utilizada encriptação simétrica com recurso ao algoritmo Advanced Encryption Standard (AES).

3 https://developer.android.com/training/articles/keystore?hl=pt-br - Acedido a 27 de Junho de 2023

4 KeyInfo https://developer.android.com/reference/android/security/keystore/KeyInfo#isInsideSecureHardware() - acedido a 05 de Julho, 2023

Listagem 65: Informação obtida sobre a chave com o alias - Authenticator


```
{
        "blockModes": [
        "GCM"
        ],
        "digests": [],
        "encryptionPaddings": [
        "NoPadding"],
        "isInsideSecureHardware": true,
        "isInvalidatedByBiometricEnrollment": false,
        "isTrustedUserPresenceRequired": false,
        "isUserAuthenticationRequired": false,
        "isUserAuthenticationRequirementEnforcedBySecureHardware": false,
        "isUserAuthenticationValidWhileOnBody": false,
        "isUserConfirmationRequired": false,
        "keyAlgorithm": "AES",
        "keySize": 128,
        "keyValidityForConsumptionEnd": null,
        "keyValidityForOriginationEnd": null,
        "keyValidityStart": null,
        "keystoreAlias": "Authenticator",
        "origin": 1,
        "purposes": 3,
        "signaturePaddings": [],
        "userAuthenticationValidityDurationSeconds": -1
}
```


Informação recolhida sobre a chave existente:

1. keystoreAlias: *Authenticator*;
2. Modo de operação: Galois/Counter Mode (GCM);
3. Chave guardada num ambiente de execução seguro : TEE;
4. Algoritmo de criptografia: AES(algoritmo de criptografia simétrica);
5. Tamanho da chave: 128 bits;

Sabendo que a aplicação faz uso da classe *Cipher*, utilizada pela *framework* Java Cryptographic Extension (JCE) para tarefas de encriptação e desencriptação, é importante consultar a sua documentação[5]. Após a sua consulta é possível conhecer os métodos e os seus atributos necessários para a criação dos *hook's* utilizados na instrumentalização.

Para observarmos o processo de criação da chave na BD local, optámos por criar uma outra conta fictícia para observar de que forma é feita a transformação do segredo inicial. Os *scripts* utilizados durante esta fase de testes foram obtidos na página de Github da *WithSecure Labs*[6]. O ficheiro `tracer-cipher.js` contém os

5 Android Developer - Cipher Class: https://developer.android.com/reference/javax/crypto/Cipher - acedido a 17 de Junho de 2023

6 https://github.com/WithSecureLabs/android-keystore-audit/blob/master/frida-scripts/ - acedido a 19 de Junho de 2023

hooks necessários para acompanhar o processo que é descrito de seguida. Podemos também ter acesso a uma cópia com alterações no apêndice B na Listagem 57.

Ao associar uma nova conta podemos observar através do resultado do *script* mostrado na figura 51, o método de encriptação utilizado e o seu correspondente vector de inicialização, com as primeiras 12 posições com o valor a "00".

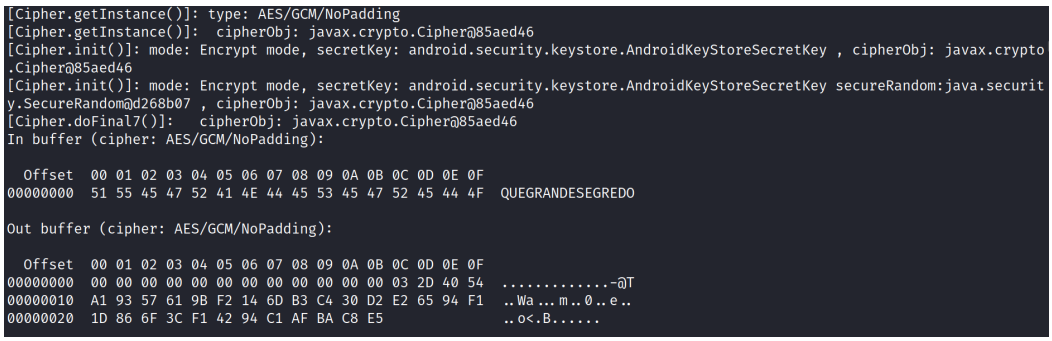


Figura 51: Hook classe Cipher: Método encriptar o segredo

Podemos observar a partir do *hexdump* obtido que o vector foi inicializado com diferentes valores nas primeiras 12 posições (00 até 0B), o restante mantém-se inalterado.

Resultado da cifragem do segredo em Hexadecimal

```
  Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 44 FB FF F4 FE 58 2F 57 B8 87 02 44 03 2D 40 54
00000010 A1 93 57 61 9B F2 14 6D B3 C4 30 D2 E2 65 94 F1
00000020 1D 86 6F 3C F1 42 94 C1 AF BA C8 E5
```

Se realizarmos a codificação com base64 do *hexdump* obtemos o valor que encontramos na BD da aplicação.

Estas 12 primeiras posições (12 bytes) correspondem ao valor do vetor de inicialização (44 FB FF F4 FE 58 2F 57 B8 87 02 44) e o seu valor é gerado de forma aleatória, e guardado em conjunto com o restante valor encriptado.

Listagem 66: Código fonte - Método de desencriptação

```
1 private final byte[] decryptInternal(byte[] bArr, byte[] bArr2) {
2         GCMParameterSpec gCMParameterSpec = new GCMParameterSpec(128, bArr, 0,
          ↪  12);
3         Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
4         cipher.init(2, this.key, gCMParameterSpec);
5         cipher.updateAAD(bArr2);
6         return cipher.doFinal(bArr, 12, bArr.length - 12);
7 }
```

Para se reverter o processo, percebemos ao ler o código fonte na Listagem 66 que o array de bytes é colocado de novo com os primeiros 12 bytes a 0 antes de ser feita a desencriptação final.

Se tal não fosse efetuado nunca seria possível reverter o processo de encriptação, tendo em atenção que o algoritmo necessita de saber quais são os bytes reservados ao vetor de inicialização e o tamanho do restante conteúdo (16 bytes - 128 bits). Esta é mais uma medida de segurança, que requer que o atacante conheça o processo de encriptação utilizado se pretender reverter o mesmo.

Não foi possível obter o conteúdo da chave utilizada no processo de criptografia. Conforme anteriormente mencionado, a chave encontra-se alojada de forma segura no TEE. É possível obter a chave caso essa esteja na *KeyStore*, mas fora do *hardware* dito seguro. Para o efeito faz-se uso do *script* apresentado na Listagem 59.

Após ler o código da listagem anterior é importante destacar que para se aceder à chave, é possível através do método `getEncoded()`, que devolve a chave no seu formato codificado original, sendo necessário apenas converter para um formato legível. Podemos obter essa informação do formato através do `getFormat()`, assim como o algoritmo que esta utiliza com o `getAlgorithm()`. Neste caso de uso foi apenas possível aceder à informação sobre o algoritmo utilizado com o método anterior, dado a restante informação não ser revelada fora do ambiente TEE.

Uma das mais valias de efetuar uma análise dinâmica, independentemente dos meios de criptografia que são utilizados, é a possibilidade de tirar partido do facto de muitas vezes a cifração ocorrer do lado do cliente. Ao controlar a sua execução podemos observar todos os mecanismos de segurança existentes e proceder à sua instrumentalização de acordo com o necessário para cada caso de uso.

Apesar de não ter sido possível obter a informação pretendida, este caso de uso serviu para aprendizagem do uso de uma ferramenta de análise dinâmica como o *Frida* e assim perceber o seu funcionamento bem como a sua potencialidade.

D E C L A R A Ç Ã O

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Técnicas de análise dinâmica a aplicações Android"*, é original e foi realizado por Luís Filipe Caixeiro Santos (2210850) sob orientação de Professor Doutor Miguel Monteiro de Sousa Frade e Professor Doutor Patrício Rodrigues Domingues.

Assinado por: **Luís Filipe Caixeiro Santos**
Num. de Identificação: 13555070
Data: 2024.05.03 22:33:25+01'00'

Leiria, Março de 2024

CHAVE MÓVEL

Luís Filipe Caixeiro Santos