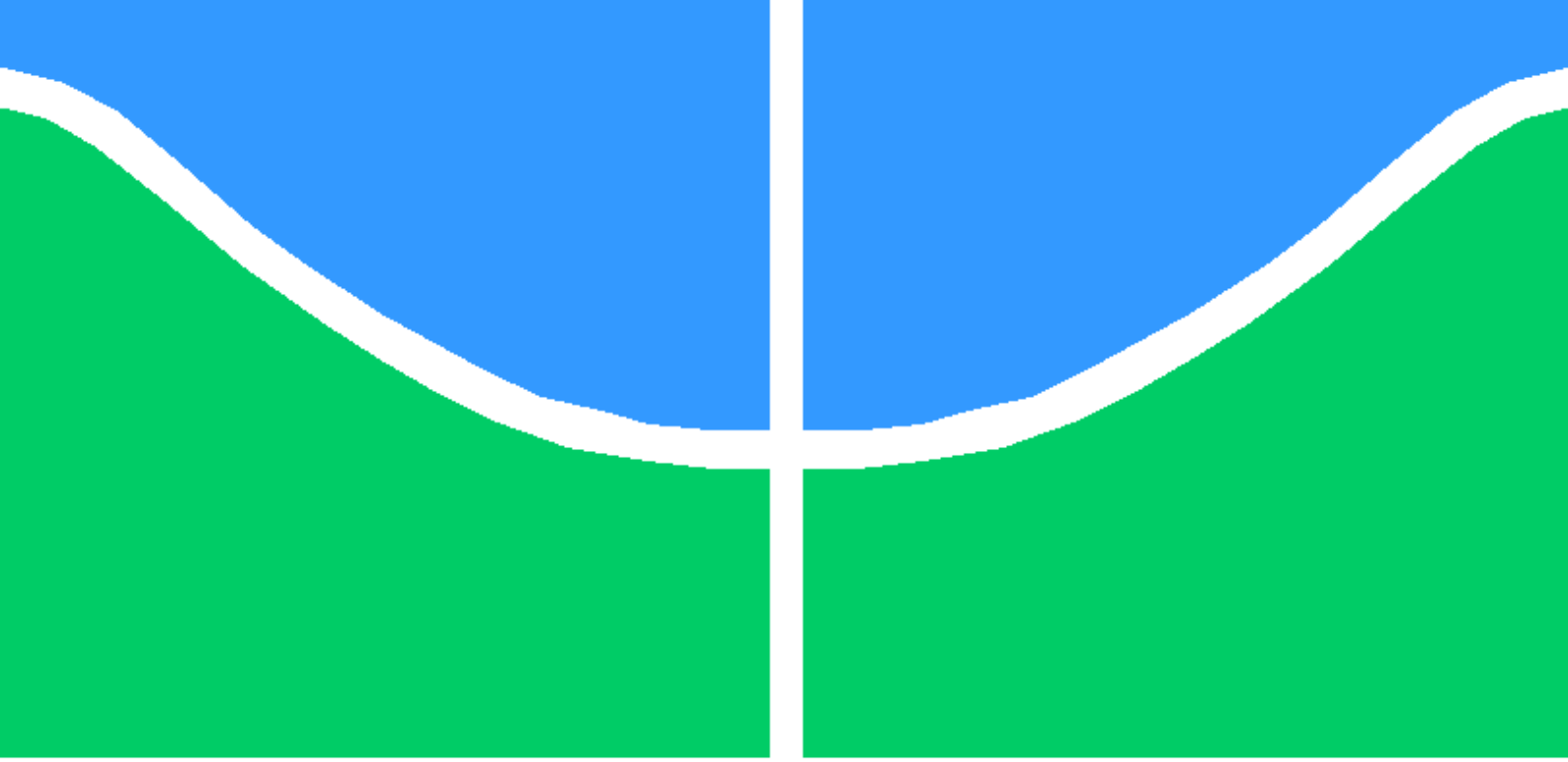


TRABALHO DE GRADUAÇÃO

ANÁLISE FORENSE EM DISPOSITIVOS COM SISTEMA OPERACIONAL ANDROID

Raquel Beatriz Silva do Nascimento

Brasília, Dezembro de 2016

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

ANÁLISE FORENSE EM DISPOSITIVOS COM SISTEMA OPERACIONAL ANDROID

Raquel Beatriz Silva do Nascimento

Relatório submetido ao Departamento de Engenharia Elétrica, como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Flávio Elias Gomes de Deus, ENE/UnB _____
Orientador

Prof. Robson de Oliveira Albuquerque, ENE/UnB _____
Co-Orientador

Fábio Lúcio Lopes de Mendonça, LabRedes/UnB _____
Examinador Interno

Dedicatória

Dedico este trabalho a todos os entusiastas do estudo de forense computacional e desejo que os mesmos possam, através deste, aumentar seu conhecimento a respeito do assunto.

Raquel Beatriz Silva do Nascimento

Agradecimentos

Agradeço primeiramente a Deus, por todas as bênçãos que Ele tem me dado. Agradeço à toda a minha família, por todo apoio dado nesses anos de graduação que não foram fáceis, principalmente meus pais, Leandro e Josélia, que sempre lutaram para que eu tivesse acesso à melhor educação possível. Agradeço também aos meus tios, Gilson e Josilene, que tiveram papel fundamental nesta jornada de graduação. Aos meus irmãos, Marcos Vinícius e Artur Vinícius que, em meio à turbulência de estudos, eram meu ponto de escape da realidade. Agradeço aos meus amigos que estão ao meu lado por tanto tempo, que sempre me deram palavras de força e estiveram presentes nos momentos alegres e tristes. A todos os professores da graduação, especialmente meus orientadores Flávio Elias e Robson Albuquerque, que souberam transmitir seus conhecimentos com dedicação e clareza. E também ao Laboratório de Forense de Dispositivos Computacionais, que proporcionou a mim o ambiente, ferramentas e tempo necessário para dedicar-me exclusivamente à realização desta pesquisa.

Raquel Beatriz Silva do Nascimento

RESUMO

Este trabalho faz a análise do desempenho de determinadas ferramentas para a forense de dispositivos móveis. A análise pericial de *smartphones* evoluiu nos últimos anos juntamente com progressos dos próprios aparelhos, cujo desempenho é aprimorado e adicionado continuamente de modo a atender as demandas dos usuários. Estes se utilizam de aparelhos celulares para diversos fins, gerando cargas de armazenamento de informações pessoais.

Foram examinadas ferramentas que possuem suporte ao sistema operacional Linux para que, ao final do trabalho, fosse criado um *Live CD* com roteiros e relatórios que tornam possíveis análises periciais. Para uma abordagem mais completa e abrangente, cenários variados foram considerados, bem como a observação de memórias voláteis e persistentes. Os resultados foram apresentados em figuras e tabelas que comparam as análises a respeito das partições examinadas dos dispositivos. Por fim, é apresentado um comparativo entre as ferramentas escolhidas e seus desempenhos.

ABSTRACT

The purpose of this project is to analyze the performances of selected open source software and applications that aim at improvements in fields of study focusing on forensic data analyzes of portable devices. Forensic science has evolved significantly over the last decades, along with the evolution of the smartphones themselves, whose features and innovations are enhanced continuously in a way to achieve the users requests and needs. The clients of so abundant technology use the devices for various purposes, which creates a demand of personal data storage and it is truly a challenge to preserve them.

Tools that provide support for Linux Operational System have been evaluated to generate a Live CD which contains templates, scripts and reports that enables possible forensic analyzes. For the sake of a more concise and embracing in-depth study, distinct scenarios have been taken into consideration, as well as researches about volatile and non-volatile memories. Results are presented as figures and tables comparing the conclusions obtained from smartphones analyzed partitions. Ultimately, comparisons among all tools contemplated in this project are introduced, as well as the benefits they can grant.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

Acrônimos

ADB	Android Debug Bridge
AFF	Advanced Forensics Format
AOT	Ahead-of-time
API	Application Program Interface
ARCH	Architecture
ARM	Advanced RISC Machine
ARP	Address Resolution Protocol
ART	Android Runtime
API	Application Programming Interface
APK	Application Package
ASCII	American Standard Code for Information Interchange
AVD	Android Virtual Device
BSD	Berkeley Software Distribution
CD	Compact Disk
CMD	Command Prompt
CSV	Comma-Separated Values
DVM	Dalvik Virtual Machine
ELF	Executable and Linking Format
ExFAT	Extended File Allocation Table
EXT2FS	Second Extended File System
EXT3FS	Third Extended File System
EXT4	Sistema de arquivos do Linux
FAT	File Alocation Table
GCC	GNU Compiler Collection
GPS	Global Positioning System
GPT	GUID Partition Table
GUID	Globally Unique IDentifier
HFS	Hierarchical File System
HTML	HyperText Markup Language
ICCID	Integrated Circuit Chip Card Identification
IDS	Intrusion Detection System

Acrônimos

IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
JPEG	Joint Photographics Experts Group
JIT	Just In Time
LiME	Linux Memory Extractor
LKM	Loadable Kernel Module
MCC	Mobile Country Code
MD5	Message-Digest algorithm 5
MMS	Multimedia Messaging Servie
MNC	Mobile Network Code
NC	Netcat
NFC	Near Field Communication
NDK	Native Development Kit
NIST	National Institute of Standards and Technology
NSRL	National Software Reference Library
NTFS	New Technology File System
RAM	Random Access Memory
RCU	Read-Copy-Update
ROM	Read-Only-Memory
PID	Process Identification
PDF	Portable Document Format
PWD	Print Working Directory
SDK	Software Development Kit
SIM	Subscriber Identity Module
SMS	Short Message Service
SO	Sistema Operacional
SQL	Structured Query Language
SSD	Solid-State Drive
TCP	Transport Control Protocol
TMPFS	Temporary Filesystem
TSK	The Sleuth Kit
UID	User Identifier
UFS	Unix File System
USB	Universal Serial Bus
URL	Uniform Resource Locator
VM	Virtual Machine
VMA	Virtual Memory Area
YAFFS2	Yet Another Flash File System
ZIP	Formato de compactação de arquivos

Capítulo 1

Introdução

Atualmente, é indiscutível o aumento do uso de *smartphones* entre a população. Na Figura 1.1 evidencia-se esse crescimento, e também apresenta-se o número de usuários destes dispositivos nos próximo anos, chegando a 2,87 bilhões em 2020. Os *smartphones* são formas compactas de computadores com alta performance, grande capacidade de armazenamento, e funcionalidades melhoradas (comparadas aos primeiros aparelhos de alguns anos atrás). Telefones celulares são os dispositivos eletrônicos mais pessoais que um usuário possui. Eles são usados para executar desde tarefas de comunicação simples, como fazer ligações e enviar mensagens de texto, até tarefas que demandam uma maior tecnologia, como navegação na Internet, uso de aplicativos de e-mail, captura de fotos e vídeos, criação e armazenamento de documentos, identificação de locais com os serviços de GPS, entre outros.

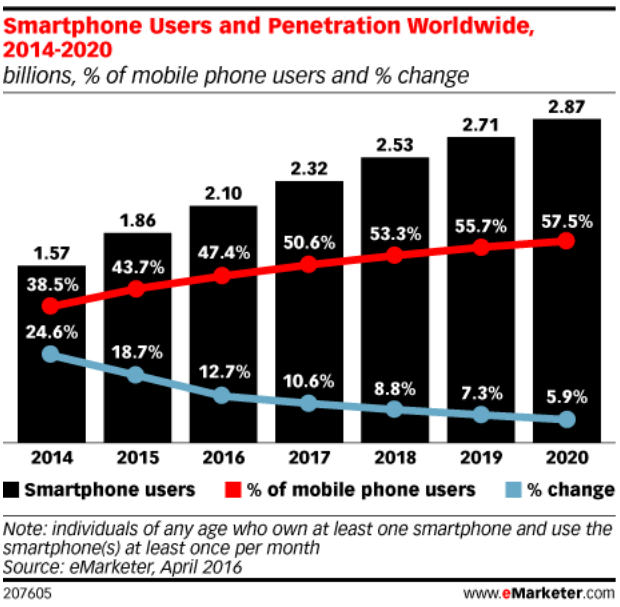


Figura 1.1: Usuários de *smartphones* e sua penetração a nível mundial, 2014-2020 [eMarketer]

1.1 Definição do Problema

A partir do momento que novos recursos e aplicações são incorporados nos telefones celulares, a quantidade de informações armazenadas nos dispositivos cresce continuamente. Assim, esses dispositivos se tornaram portadores de dados pessoais, armazenando as atividades de um usuário. Com o aumento da prevalência dos celulares na vida diária das pessoas em geral, dados adquiridos de celulares têm se tornado uma fonte inestimável de evidência para investigações relativas a processos criminais, civis e até mesmo casos de alta importância. A área que estuda a recuperação de evidências digitais de telefones celulares é chamada de **forense de dispositivos móveis**.

Esse crescimento do uso de *smartphones* faz com que seja necessário uma análise minuciosa destes dispositivos, principalmente daquele que é predominante no cenário mundial, o Android [Gartner]. Este sistema operacional tem várias versões e características específicas que tornam sua análise essencial na descoberta de dados de usuários.

1.2 Objetivos

O Objetivo Geral deste trabalho é a análise forense utilizando ferramentas *open source* para obtenção de dados de dispositivos móveis, com sistema operacional Android, para que, ao final seja feita uma comparação das mesmas e dos tipos de dados extraídos. Para se chegar a este objetivo, são propostos os seguintes tópicos, que são os Objetivos Específicos:

- Descrição dos principais conceitos que envolvem a forense de dispositivos móveis e o sistema operacional Android;
- Resultado da análise dos dispositivos conforme cada ferramenta utilizada;
- Análise da utilidade, praticidade e retorno de cada ferramenta utilizada;
- Criação de um *Live CD* em que seja possível usar as ferramentas analisadas, além de conter roteiros que orientem a análise forense dos dispositivos.

1.3 Justificativa

É raro conduzir uma investigação sem que algum dispositivo móvel esteja envolvido como fonte de evidências, por isso é importante que se estude acerca das técnicas forenses relacionadas a estes telefones celulares. Além disso, é essencial que seja estudado o sistema operacional Android especificamente, porque hoje é aquele que tem mais expressividade no mercado [Gartner], como pode-se notar na Tabela 1.1.

Tabela 1.1: Vendas mundiais de *smartphones* para usuários finais por Sistemas Operacionais até o final do primeiro semestre de 2016 - Milhares de Unidades [Gartner]

Sistema Operacional	2Q16 Unidades	2Q16 - Quota de Mercado (%)	2Q15 Unidades	2Q15 - Quota de Mercado (%)
Android	296,912.8	86.2	271,647.0	82.2
iOS	44,395.0	12.9	48,085.5	14.6
Windows	1,971.0	0.6	8,198.2	2.5
Blackberry	400.4	0.1	1,153.2	0.3
Outros	680.6	0.2	1,229.0	0.4
Total	344,359.7	100.0	330,312.9	100.0

Com relação aos fabricantes de telefones celulares, é importante que sejam analisados aqueles que são mais expressivos para o consumidor. No caso daqueles que possuem o sistema operacional Android, o fabricante analisado neste trabalho será a Samsung. Como apresentado na Tabela 1.2, a Samsung foi a que mais vendeu dispositivos móveis, até o final do primeiro semestre de 2016 [Gartner].

Tabela 1.2: Vendas mundiais de *smartphones* para usuários finais por Fabricantes até o final do primeiro semestre de 2016 - Milhares de Unidades [Gartner]

Fabricantes	2Q16 Unidades	2Q16-Quota de Mercado (%)	2Q15 Unidades	2Q15-Quota de Mercado (%)
Samsung	76,743.5	22.3	72,072.5	21.8
Apple	44,395.0	12.9	48,085.5	14.6
Huawei	30,670.7	8.9	26,454.4	8.0
Oppo	18,489.6	5.4	8,073.8	2.4
Xiaomi	15,530.7	4.5	15,464.5	4.7
Outras	158,530.3	46.0	160,162.1	48.5
Total	344,359.7	100.0	330,312.9	100.0

Existem várias ferramentas que realizam a captura e a análise de informações de um *smartphone*, porém, a maioria destas são *softwares* com foco comercial. Alguns exemplos destes *softwares*, são a *Cellebrite* que é uma das mais utilizadas por ser a mais completa e com uma grande cobertura de telefones celulares e também o *FTK Imager*, que possui tanto versões com foco comercial, quanto versões gratuitas.

Tendo em vista este foco comercial, é importante que se analise ferramentas *open source* que estejam ao alcance de qualquer pessoa que deseje realizar uma perícia em dispositivos móveis. Além disso, ferramentas deste tipo auxiliam na redução de custo de uma análise, além de permitir a criação, por exemplo, de uma comunidade para discussão do tema. A criação de um *Live CD* facilita os processos de instalação de ferramentas, já que as mesmas estariam prontas para o uso do investigador, além dos roteiros que auxiliariam na aprendizagem quanto ao uso destas e nos

resultados finais das análises.

Três versões do Android serão examinadas para que haja uma ampla cobertura amostral do real cenário. A Figura 1.2 mostra dados coletados durante um período de 7 dias acabando no dia 7 de Novembro de 2016 [Versions], que mostram o número relativo de dispositivos executando uma determinada versão da plataforma Android. As versões que serão analisadas neste trabalho, serão a 4.0.3 - *Ice Cream Sandwich*, a 5.0.1 - *Lollipop* e 6.0 - *Marshmallow*. A versão 4.0.3 será abordada para que seja feita uma comparação a níveis arquiteturais, já que esta versão utilizava o *Dalvik VM*. A versão 5.0.1 foi uma atualização realizada pelo Androiid, para corrigir alguns *bugs* da versão 5.0, incluía-se a resolução de problemas com o *playback* de vídeos e a correção de falhas de senhas. Enquanto a 6.0 foi escolhida para que fosse analisada uma versão mais recente do Android e com significativa representatividade no mercado.

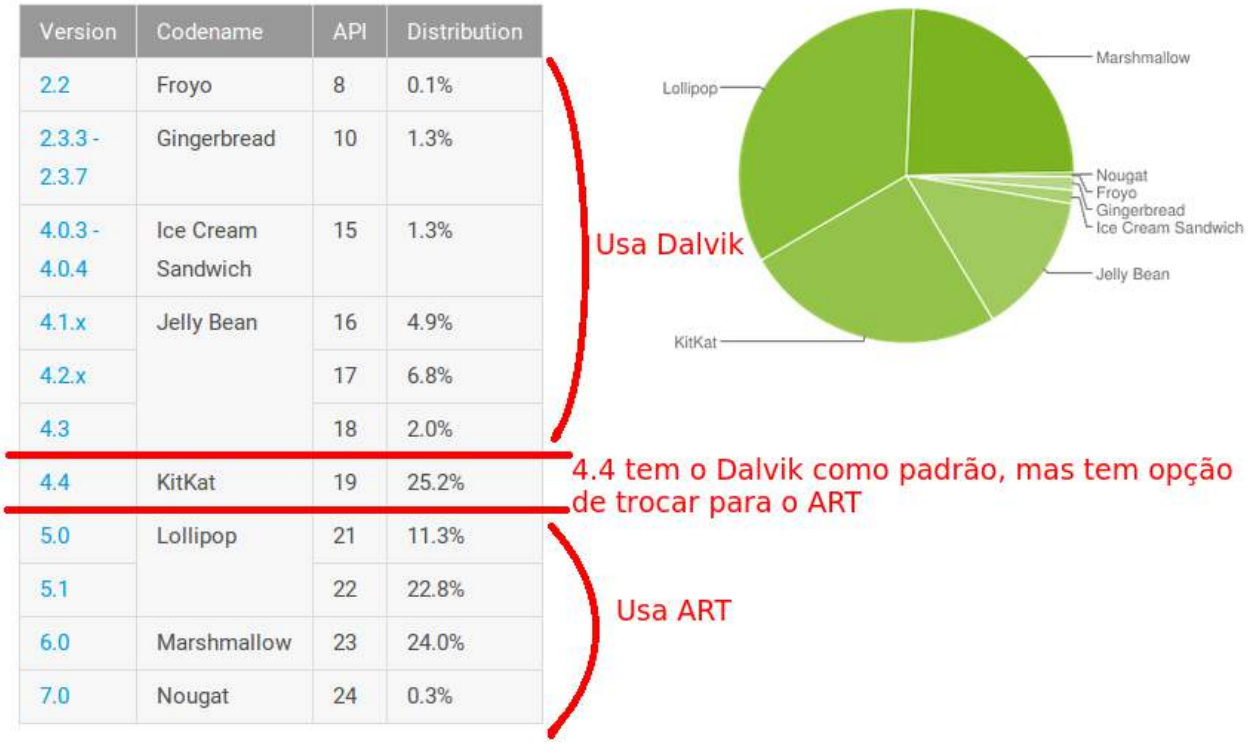


Figura 1.2: Distribuição de versões do Android em dispositivos móveis [Versions]

Além disso, no cenário com a versão 5.0.1 que trata de um cenário real, serão analisados além de informações essenciais como registros de chamadas, agenda de contatos, SMS, e-mail, entre outros, informações acerca de aplicativos muito utilizados no dia a dia de usuários de *smartphones*. Os aplicativos escolhidos para serem analisados, foram baseados nas informações da Figura 1.3, que dizem respeito aos *apps* mais utilizados pelos brasileiros em 2015 [IBOPE].

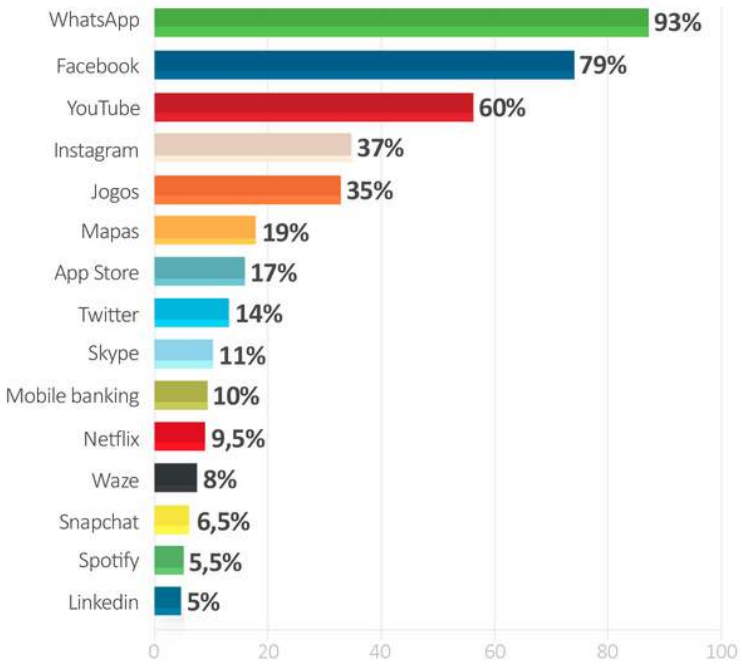


Figura 1.3: Aplicativos mais utilizados pelos brasileiros em 2015 [IBOPE]

1.4 Estrutura do Trabalho

No Capítulo 2, Fundamentação Teórica, serão abordados os principais fundamentos utilizados nesse projeto, incluindo aqueles relacionados à forense de dispositivos móveis e ao sistema operacional Android.

No Capítulo 3, os métodos e as ferramentas, com suas principais características, são apresentados. Cada seção será responsável pela descrição de cada *software* utilizado.

No Capítulo 4, serão feitas análises com as ferramentas apresentadas no Capítulo 3 em três cenários diferentes, além de apresentar comparações, a partir dos resultados obtidos e instruções de como foi criado o *Live CD*.

Por último, o Capítulo 5, contém as conclusões acerca das análises feitas. Será feito um resumo em que se verá quais são as melhores ferramentas para certos tipos de situações que podem ser encontradas em uma investigação.

Capítulo 2

Fundamentação Teórica

Esse capítulo apresenta os principais fundamentos teóricos nos quais esse trabalho foi embasado. Os conceitos e mecanismos de funcionamento dos recursos utilizados ao longo das análises são detalhados.

2.1 Forense em Dispositivos Móveis

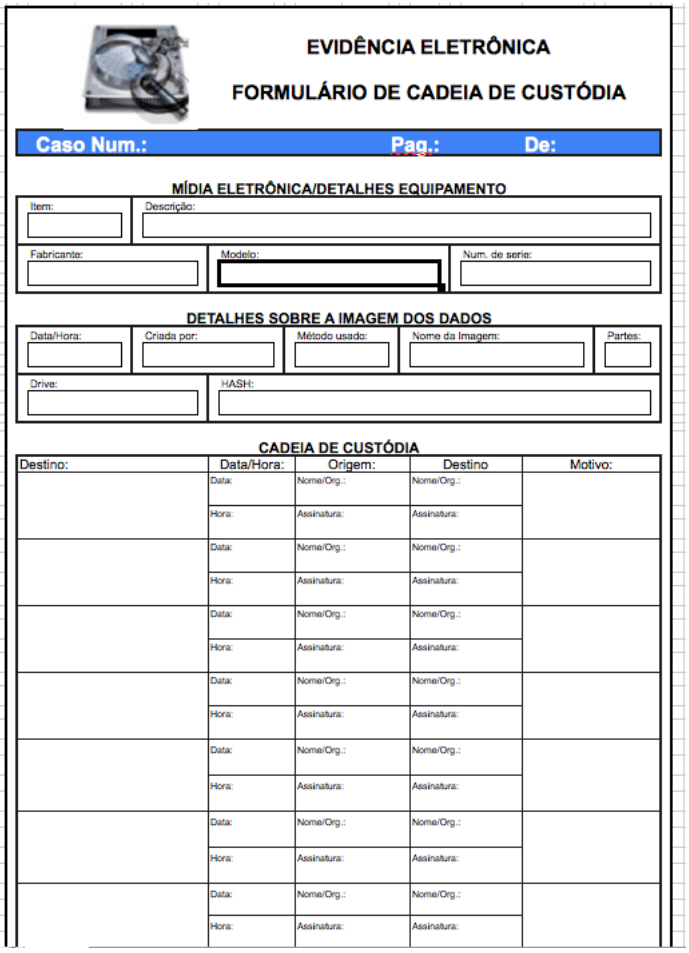
A prática forense em dispositivos móveis faz parte da grande categoria da forense digital. Trata da extração, recuperação e análise de evidências digitais ou dados de um dispositivo móvel, sobre condições forenses específicas. Simplificando, trata do acesso de dados armazenados em dispositivos, o que inclui SMS, contatos, registros de chamadas, fotos, vídeos, documentos, arquivos de aplicativos, históricos de navegadores, entre outros, e também trabalha na recuperação de dados deletados dos dispositivos, usando algumas técnicas forenses.

2.1.1 Procedimentos e Cuidados

É importante que o processo de recuperação ou o acesso a detalhes de um dispositivo seja feito com todos os princípios forenses básicos, se os dados forem utilizados em uma corte e para manter a integridade do mesmo. Esses princípios devem seguir a cadeia de custódia, que é o processo de documentar a história cronológica da evidência. A Figura 2.1 demonstra um modelo de formulário de cadeia de custódia a ser preenchido pelo investigador, para que a atividade forense seja realizada de maneira íntegra, sem modificação ou perda dos dados.

Além de se respeitar os princípios da cadeia de custódia, preenchendo-se o formulário descrito anteriormente, é preciso que alguns cuidados sejam tomados e estes serão determinados em algumas etapas [Simão 2011]:

- **Apreensão**: nesta fase, além de documentar tudo o que foi encontrado com o celular, como baterias externas e cartões de memória, é importante que o perito esteja presente no momento para as primeiras verificações do celular. É preciso observar se o celular estava ligado ou



EVIDÊNCIA ELETRÔNICA

FORMULÁRIO DE CADEIA DE CUSTÓDIA

Caso Num.: Pag.: De:

MÍDIA ELETRÔNICA/DETALHES EQUIPAMENTO

Item:	Descrição:		
Fabricante:	Modelo:	Num. de serie:	

DETALHES SOBRE A IMAGEM DOS DADOS

Data/Hora:	Criada por:	Método usado:	Nome da Imagem:	Partes:
Drive:	HASH:			

CADEIA DE CUSTÓDIA

Destino:	Data/Hora:	Origem:	Destino	Motivo:
	Data:	Nome/Org.:	Nome/Org.:	
	Hora:	Assinatura:	Assinatura:	
	Data:	Nome/Org.:	Nome/Org.:	
	Hora:	Assinatura:	Assinatura:	
	Data:	Nome/Org.:	Nome/Org.:	
	Hora:	Assinatura:	Assinatura:	
	Data:	Nome/Org.:	Nome/Org.:	
	Hora:	Assinatura:	Assinatura:	
	Data:	Nome/Org.:	Nome/Org.:	
	Hora:	Assinatura:	Assinatura:	
	Data:	Nome/Org.:	Nome/Org.:	
	Hora:	Assinatura:	Assinatura:	
	Data:	Nome/Org.:	Nome/Org.:	
	Hora:	Assinatura:	Assinatura:	

Figura 2.1: Modelo de formulário de cadeia de custódia [Neukamp]

desligado, se existe algum mecanismo de proteção na tela de bloqueio, entre outros. Outra ação a ser tomada, é a de isolar o aparelho da rede de telefonia, pois se o celular puder receber mensagens, ligações, isto pode alterar o curso das investigações. O mais recomendado é que o celular seja colocado no modo avião ou em um recipiente que bloqueie os sinais, até que o perito esteja no seu local de análise. No caso de o aparelho estar ligado e, além disso, estiver com bloqueio de tela, o investigador pode perguntar diretamente para o dono do celular a senha do mesmo, em casos extremos é necessário o uso da força bruta para que seja descoberta o padrão de desbloqueio do dispositivo. Todo este processo tem de ser documentado para o caso de possíveis dúvidas posteriores;

- **Aquisição de dados**: esta fase é a que precisa de mais precisão técnica por parte do perito, pois a extração dos dados tem de ser feita de forma que não altere os dados e de que não seja perdida nenhuma informação importante. O técnico precisa ser especialista com conhecimentos específicos em sistemas Android, pois provavelmente será necessária uma intervenção manual por parte do perito. Também deve preocupar-se em extrair informações com o mínimo de intervenção do sistema. Primeiramente, se o celular tiver um cartão de memória externo, deve haver uma preocupação em fazer uma cópia do mesmo para que seja

analisado posteriormente. Esta cópia pode ser realizada através de algumas ferramentas forenses e pode ser gerado o *hash* destes dados para que seja comparado depois, original e cópia. É preferível que o perito retorne ao celular, a cópia do cartão de memória, deixando o original armazenado em um lugar seguro em que não sofrerá nenhum dano. Depois disso, se o celular já não estiver isolado da rede, deve-se isolá-lo para que, como descrito anteriormente, não haja modificação alguma dos dados. No caso em que o celular não contenha algum tipo de controle de acesso, o perito então pode começar a investigação dos dados do celular. Primeiramente, deve-se verificar as concessões de permissões, se o aparelho estiver com permissões de super usuário ativadas, o processo de espelhamento da partição do sistema pode ser realizado. Após o espelhamento, o processo de extração em si pode ser iniciado. Neste processo, os dados que serão coletados, deverão ser copiados para o cartão de memória do perito que foi inserido anteriormente, logo após a cópia do cartão de memória original. Para realizar a extração dos dados, é necessário que seja instalado um ou mais aplicativos no smartphone. Esta instalação se dá por meio do Android Debug Bridge - ADB e o dispositivo tem de estar com o modo de depuração ativado. A fim de complementar as análises, o perito pode comparar os dados coletados, com os dados originais no próprio celular. No caso em que o celular tenha controle de acesso (padrão de senha na tela de bloqueio), o perito pode tentar obter acesso via conexão USB com sua estação de trabalho e então tentar acessá-lo via ADB. Se o acesso ADB não estiver liberado, o que seria mais provável, só resta ao perito a utilização de técnicas de força bruta para tentar descobrir a senha de acesso (estas serão discutidas posteriormente no Capítulo 4). Além disso, assim como no outro caso, é possível pelo menos fazer a análise do cartão de memória, se este existir. Tudo tem de ser documentado a cada etapa, para que não haja controvérsias ou queixas no momento da apresentação dos resultados da perícia. Até mesmo informações como versão e *kernel* do Android, cor e modelo do aparelho celular, têm de ser documentadas.

- **Análise**: nesta fase, o perito, primeiramente, deve estar atento e definir os objetivos do exame. Dependendo do caso, ele pode focar somente nas imagens e vídeos ou somente nas ligações do usuário do celular. Após esta definição, deve-se encontrar informações para individualizar o aparelho, como, por exemplo, contas de e-mail que provem que o usuário principal do mesmo é o suspeito. A primeira análise feita, deve ser a do cartão memória externo que foi copiado anteriormente e, depois, é feita a análise da memória interna do celular que foi espelhada. Assim como nas outras fases, a documentação completa deve ser realizada.

- **Geração do relatório final**: essa fase nada mais é, como o nome já informa, a geração de um relatório final, com as técnicas utilizadas e informações coletadas durante as outras fases. É essencial, pois, se algo for esquecido, pode ser contestado posteriormente de alguma maneira.

A Figura 2.2 exemplifica as etapas descritas anteriormente. Seguindo estas melhores práticas, o perito pode extrair os dados necessários de forma segura. E para que tais fases sejam concluídas com sucesso, algumas técnicas são necessárias para que as informações sejam coletadas sem problemas.

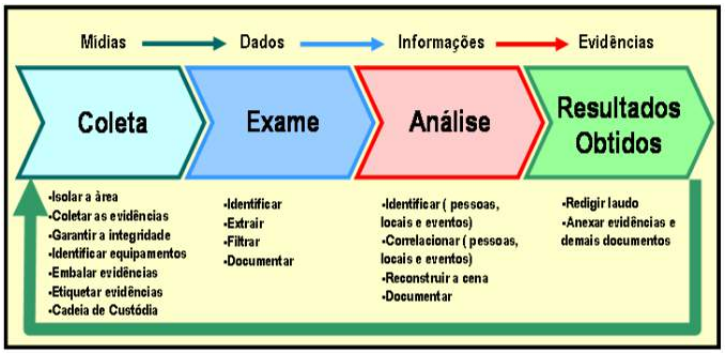


Figura 2.2: Etapas da análise forense de um dispositivo móvel [Santos]

2.1.2 Tipos de extração de dados

O item **aquisição de dados** citado acima, pode ser caracterizado em vários tipo de extrações. A Figura 2.3 mostra uma pirâmide na qual, à medida que se avança ao seu topo, mais tecnologia, mais conhecimento técnico, mais recursos financeiros e, usualmente, mais tempo são necessários para realizar a extração dos dados armazenados.

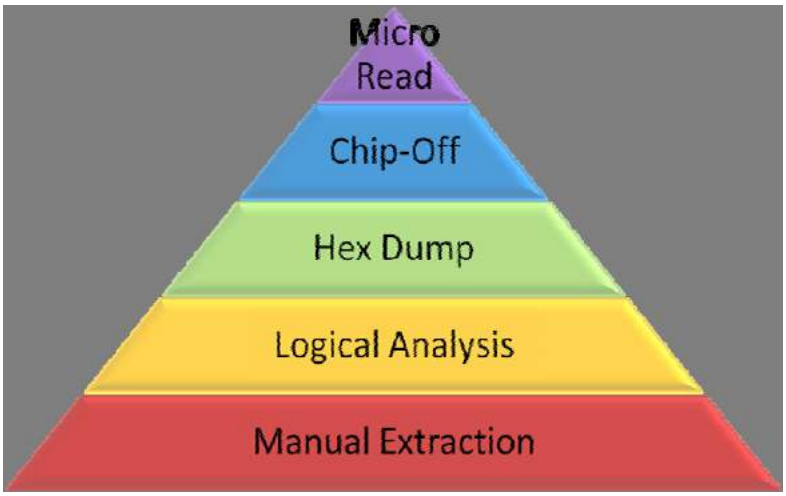


Figura 2.3: Sistema de classificação de ferramentas de dispositivos móveis [Murphy]

A **extração manual**, que é a base da pirâmide, consiste na constatação manual dos vestígios através da manipulação do aparelho em exame. Esse método não requer conhecimentos avançados, pois, basicamente, o perito utiliza o dispositivo como um usuário comum. Os dados podem ser transcritos manualmente ou a tela do aparelho pode ser fotografada. Obviamente, informações apagadas não podem ser recuperadas com este método. Essa forma de extração deve ser utilizada apenas no último caso, pois ela é suscetível a uma série de problemas. Há o risco da alteração acidental dos dados ou erro humano na transcrição, visto tratar-se de um processo tedioso e que consome muito tempo [Velho 2016].

O próximo nível é o da **extração lógica** que requer utilização de ferramentas forenses es-

pecíficas, nas quais os equipamentos portáteis devem ser conectados para viabilizar a extração automática dos dados de sua memória interna ou externa. A conexão pode se dar por cabos, ou por comunicação sem fio. As ferramentas forenses enviam uma série de comandos ao aparelho em exame, utilizando as APIs proprietárias de cada fabricante, e o aparelho responde fornecendo os dados requisitados. As respostas são coletadas pelas ferramentas forenses que geram relatórios dos dados extraídos. Essa forma de extração implica que dados apagados não são recuperados. Uma variante avançada da extração lógica é a extração do sistema de arquivos do aparelho (*File System Dump*). O funcionamento é similar, mas as ferramentas forenses utilizam um conjunto diferente de APIs do SO do dispositivo para ter acesso às suas partições e aos seus arquivos internos. Nessa modalidade, registros excluídos dos bancos de dados que ainda não foram sobrepostos são passíveis de análise, possibilitando a recuperação de algumas informações já apagadas do aparelho [Velho 2016].

A camada ***Hex Dump*** da pirâmide diz respeito à extração física de dados. Nesse nível, as ferramentas forenses conseguem acesso direto ao conteúdo da memória *flash* dos dispositivos, funcionando como a cópia *bit* a *bit* dos exames de mídia de armazenamento convencionais. O grande desafio consiste na decodificação dos dados e reconstrução dos diversos sistemas de arquivos, principalmente no caso de sistemas proprietários não documentados pelos fabricantes. Por outro lado, a recuperação de dados apagados é maximizada por causa do acesso aos blocos não alocados de memória [Velho 2016].

As camadas ***Chip-Off*** e ***Micro Read*** dizem respeito às técnicas mais avançadas de extração e requerem conhecimentos em eletrônica por parte do perito, já que requerem a remoção física do circuito integrado de memória da placa de circuito impresso. Na técnica *Chip-Off*, após removido, o circuito de memória é lido *bit* a *bit* em um equipamento apropriado. O grande problema é que os circuitos de memória NAND utilizam um algoritmo de *"wear levelling"*, o mesmo utilizado em SSDs para maximizar a sua vida útil, e esse algoritmo precisa sofrer engenharia reversa para que os dados binários brutos obtidos possam ser colocados em ordem lógica. A técnica mais avançada de é a *Micro Read*, que consiste na leitura em microscópio eletrônico do estado de cada porta lógica do circuito integrado de memória. A utilização dessa técnica requer um time de especialistas, equipamentos apropriados e um profundo conhecimento das particularidades do *hardware* em questão, sendo somente justificável em casos críticos envolvendo segurança nacional, e somente se as técnicas anteriores tivessem falhado [Velho 2016].

2.2 O Sistema Operacional Android

O sistema operacional Android é um SO *open source* utilizado em dispositivos móveis. Foi escrito primeiramente em *Java* e é baseado no sistema operacional Linux, desenvolvido inicialmente pela Android Inc. e posteriormente comprada pela Google em 2005. Baseado em uma versão modificada do *kernel* 2.6 do Linux, o código do Android foi realizado pela Google sobre uma licença *Apache* que também é um *software open source*.

2.2.1 A Arquitetura da Plataforma Android

A arquitetura consiste em uma pilha de camadas que são executadas uma acima da outra. A arquitetura do Android pode ser compreendida quando se observa o que estas camadas são e o que elas fazem. Apresenta-se na forma de uma pilha de *software*, em que se compreende o *kernel*, bibliotecas, ambiente de execução, aplicações, *middleware*, e serviços. Cada camada da pilha (e também os elementos dentro de cada camada) é integrada de maneira que seja provida um ambiente de execução ótimo para dispositivos móveis. A Figura 2.4 mostra a arquitetura utilizada até a versão 4.4 - *Kit Kat* do Android, a partir da versão 5.0 - *Lollipop* houve uma mudança significativa no modo de execução e a nova arquitetura é apresentada na Figura 2.5. A seguir será apresentada uma visão de cada camada da arquitetura.

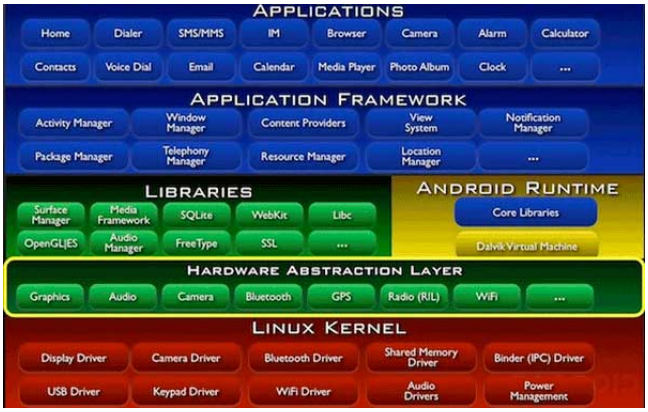


Figura 2.4: Arquitetura do Android até a versão 4.4 - *Kit Kat*

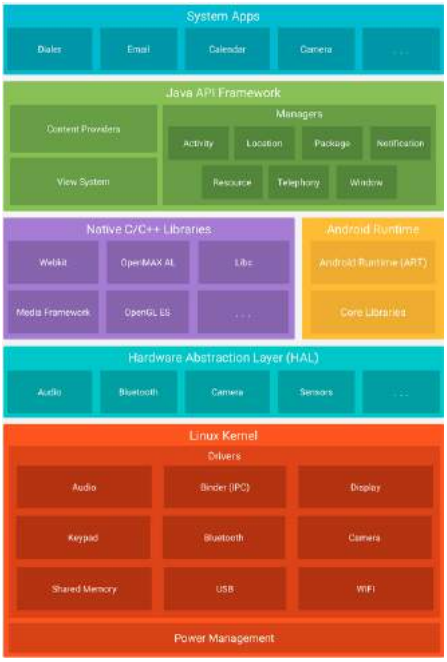


Figura 2.5: Arquitetura do Android a partir da versão 5.0 - *Lollipop*

O *kernel* do SO Android é baseado no *kernel* do Linux com algumas mudanças arquiteturais feitas pelo Google. O SO Linux foi escolhido por ser uma plataforma portável que pode ser compilada facilmente em diferentes tipos de *hardware*. O *kernel* do Linux é posicionado na base da pilha de *softwares* e provê um nível de abstração entre o *hardware* do dispositivo e as camadas superiores.

As funcionalidades de *core* do Android, como gerenciamento de processos, memória, segurança e rede, são administrados pelo *kernel* do Linux. O Linux é uma plataforma comprovada, quando se trata de segurança e gerenciamento de processos. Cada versão do Android possui uma versão diferente de *kernel*. As versões específicas são determinadas de acordo com cada dispositivo e seu respectivo *chipset*.

Acima do *kernel* se encontram as bibliotecas nativas do Android. É com a ajuda destas bibliotecas que o dispositivo lida com diferentes tipos de dados. As bibliotecas são escritas nas linguagens de programação C ou C++ e são específicas para um *hardware* particular. *Surface Manager*, *Media framework*, *SQLite*, *OpenGL*, entre outras são algumas das bibliotecas nativas mais importantes.

Aplicações para Android são programadas usando-se a linguagem *Java* de programação. A principal razão para essa escolha é por ser uma linguagem conhecida e por ter uma grande base de desenvolvimento. O Android quis ter vantagem desta comunidade de desenvolvimento já existente, ao invés de criar uma nova linguagem.

Quando um programa *Java* é compilado, este resulta em um *bytecode* que pode ser executado com o auxílio de uma *Java Virtual Machine* - JVM. No caso do Android, este *bytecode* é posteriormente convertido para *bytecode Dalvik* por um compilador *dex*. Este novo código é então colocado em uma *Dalvik Virtual Machine* - DVM que pode ler e usar o mesmo. Assim, os arquivos *.class* do compilador *Java* são convertidos em arquivos *.dex* usando a ferramenta *dx*. O *bytecode Dalvik* é um *bytecode* otimizado adequado para ambientes de *low-memory* e de *low-processing*. Também percebe-se que o *bytecode* usado pela JVM consiste em um ou mais arquivos *.class*, dependendo do número de arquivos *Java* que estão presentes em uma aplicação, mas o *bytecode Dalvik* é composto de apenas um arquivo *.dex*. Cada aplicação presente no Android executa sua própria instância da DVM.

Desde a versão 5.0 - *Lollipop* do Android, como foi mostrado na Figura 2.5, *Dalvik* foi substituído pelo *Android Run Time* - ART como plataforma padrão. ART foi introduzido na versão 4.4 - *Kit Kat* de modo experimental. *Dalvik* usa compilação *just-in-time* - JIT, esta compila o *bytecode* toda vez que uma aplicação é executada. Porém, ART usa compilação *ahead-of-time* - AOT, que é executada no momento em que uma aplicação é instalada. Esta mudança reduz o uso do processador do dispositivo móvel, já que o processamento que uma compilação utiliza de um ponto de vista geral, durante as operações de uma aplicação, é reduzido.

A camada *application framework* apresenta os serviços que atuam como gerenciadores das aplicações executadas no Android. É responsável pela performance de várias funções cruciais como gerenciamento de recursos, gerenciamento de chamadas, e assim por diante. Alguns desses serviços chaves são [Tamma e Tindall 2015]:

- **Provedor de conteúdo**: permite que aplicações publiquem e compartilhem dados com outras aplicações;
- **Visualizador do sistema**: provê um conjunto de visualizações extensíveis usados para criar interfaces de usuários de aplicativos;
- **Gerenciador de atividades**: controla todos os aspectos do ciclo de vida de aplicações e pilha de atividades;
- **Gerenciador de recursos**: provê acesso a recursos incorporados não codificados, como *strings*, configurações de cores, e *layouts* de interface de usuário;
- **Gerenciador de notificações**: permite que aplicações exibam alertas e notificações para o usuário;
- **Gerenciador de pacotes**: o sistema pelo qual aplicações são capazes de encontrar informações sobre outras aplicações que atualmente estão instaladas no dispositivo;
- **Gerenciador de telefonia**: provê informações para a aplicação sobre serviços de telefonia disponíveis no dispositivo, como o *status* e informações sobre assinantes;
- **Gerenciador de localização**: provê acesso a serviços de localização permitindo que uma aplicação receba atualizações sobre mudanças.

A camada mais elevada na pilha da arquitetura do Android, consiste nos aplicativos (chamados de *apps*) que são programas nos quais os usuários interagem diretamente. Existem dois tipos de *apps* a serem discutidos [Tamma e Tindall 2015]:

- ***Apps* do sistema**: estes são aplicativos que são pré instalados no dispositivo e são adquiridos no momento da compra do mesmo. Aplicativos como navegador padrão, cliente de e-mail, contatos, entre outros, são exemplos de *apps* do sistema. Estes não podem ser desinstalados ou modificados pelo usuário. Geralmente estão presentes no diretório */system*. Até a versão 4.4 - *Kit Kat* do Android, todos os *apps* presentes em */system* eram tratados igualmente. Porém, a partir da versão 4.4 - *Kit Kat* em diante, *apps* instalados em */system/priv-app/* são tratados como aplicativos com privilégios, e permissões são concedidas com nível de proteção apenas para os *apps* com privilégios;
- ***Apps* instalados pelo usuário**: estes são os aplicativos que os usuários realizam o *download* e instalação a partir de várias plataformas de distribuição, como o *Google Play*. *Google Play* é a loja oficial de aplicativos para dispositivos com o sistema operacional Android, nele os usuários podem procurar e fazer o *download* dos aplicativos. Estes estão presentes no diretório */data*.

2.2.2 Permissões de Aplicativos na Plataforma Android

O conjunto das permissões dos aplicativos presentes em um dispositivo com SO Android, é definido no arquivo *AndroidManifest.xml*. Para acessar recursos presentes no aparelho celular,

cada *app*, seja ele nativo do sistema ou não, deve requisitar permissões no seu específico arquivo *AndroidManifest.xml*. O gerenciamento das permissões de todos os aplicativos é feito no arquivo *packages.xml*.

Antes da versão 6.0 - *Marshmallow* do Android, as permissões eram dadas aos aplicativos no momento da instalação do mesmo. Eram listados, todos os tipos de recursos aos quais iria se obter acesso. Porém, a partir desta versão, os recursos são requisitados no momento em que o *app* precisar. A Figura 2.6 mostra esta diferença, à esquerda, as permissões são concedidas no momento em que o *app* foi instalado e à direita é o novo sistema em que o usuário só concede a permissão no momento de uso do recurso.

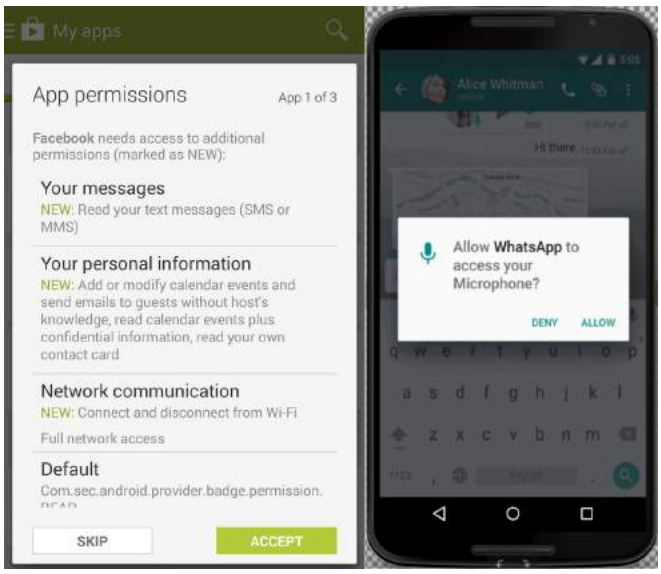


Figura 2.6: Diferença entre os tipos de concessão de permissões aos aplicativos [Chirgwin] [Eason]

Alguns níveis de proteção são determinados no arquivo *AndroidManifest.xml*, eles se referem ao nível de intenção de uso de determinado recurso e também a consequência da utilização da mesma. Estes são apresentados a seguir [Wei et al. 2012]:

- ***Normal***: permissões que apresentam risco mínimo para *apps* e são concedidas automaticamente pela plataforma Android sem a necessidade da aprovação explícita do usuário;
- ***Dangerous***: permissões que fornecem acesso aos dados pessoais sensíveis do usuário e vários recursos do dispositivo. As permissões podem ser concedidas ou não pelo usuário, a forma como a requisição irá aparecer no *smartphone*, dependerá da versão do Android, como já foi explicitado na Figura 2.6;
- ***Signature***: permissões que significam o privilégio mais alto; elas só podem ser obtidas se o aplicativo que estiver solicitando-a for assinado com o certificado do fabricante do dispositivo;
- ***signatureOrSystem***: permissões que só são concedidas a aplicativos que estão na imagem do sistema Android ou são assinados com o mesmo certificado na imagem do sistema. As

permissões nesta categoria são usadas para certas situações especiais, em que vários fornecedores têm aplicativos incorporados em uma imagem do sistema e precisam compartilhar recursos especiais específicos porque eles estão sendo criados juntos.

O Android também define um conjunto de categorias de permissão com base na funcionalidade. No total, há 11 categorias, com nomes auto-explicativos [Wei et al. 2012]: *Cost Money* (custa dinheiro), *Message* (mensagens), *Personal Info* (informações pessoais), *Location* (localização), *Network* (rede), *Accounts* (contas), *Harware Controls* (controles de hardware), *Phone Calls* (chamadas telefônicas), *Storage* (armazenamento), *System Tools* (ferramentas de sistema) e *Development Tools* (ferramentas de desenvolvimento). Há também uma categoria padrão que é usada quando nenhuma categoria é especificada na definição de uma permissão do Android.

Tabela 2.1: Permissões do nível *Dangerous* e grupos de permissão [Permissions]

Grupos de Permissão	Permissões
Calendário	-*READ_CALENDAR* -*WRITE_CALENDAR*
Câmera	-*CAMERA*
Contatos	-*READ_CONTACTS* -*WRITE_CONTACTS* -*GET_ACCOUNTS*
Localização	-*ACCESS_FINE_LOCATION* -*ACCESS_COARSE_LOCATION*
Microfone	-*RECORD_AUDIO*
Telefone	-*READ_PHONE_STATE* -*CALL_PHONE* -*READ_CALL_LOG* -*WRITE_CALL_LOG* -*ADD_VOICEMAIL* -*USE_SIP* -*PROCESS_OUTGOING_CALLS*
Sensores	-*BODY_SENSORS*
SMS	-*SEND_SMS* -*RECEIVE_SMS* -*READ_SMS* -*RECEIVE_WAP_PUSH* -*RECEIVE_MMS*
Armazenamento	-*READ_EXTERNAL_STORAGE* -*WRITE_EXTERNAL_STORAGE*

A Tabela 2.1 mostra uma descrição das permissões que são consideradas de nível *Dangerous*. Estas permissões precisam da autorização do usuário para que sejam concedidas ao aplicativo.

2.2.3 A Plataforma Android e sua Análise Forense

Todos os aplicativos utilizados por um usuário de *smartphone* podem ser utilizados por um perito forense, pois armazenam dados do usuário. Este armazenamento de informações é feito em dois locais: interno e externo. O armazenamento externo dos dados, por um *Secure Digital Card* (cartão SD), que geralmente é formatado com o sistema de arquivos *Microsoft FAT32*. Dependendo do dispositivo, o cartão pode ser removido podendo ser copiado para posteriores análises e se a extração não for possível, a imagem do cartão deve ser feita. Outros dados podem ser armazenados internamente em um *chip* de memória *flash*. A memória interna também armazena arquivos de sistema e é gerenciado pela *Application Program Interface* - API do Android. Quando novos aplicativos são instalados, cria-se uma pasta para o *app* no subdiretório */data/data* [Lellis e Cruz 2015].

Outra característica importante é o Android Software Development Kit (SDK) que é um conjunto de ferramentas usadas para o desenvolvimento de aplicações para a plataforma Android. O SDK do Android inclui as seguintes especificações: bibliotecas requeridas pelas aplicações, um *debugger* (ADB), um emulador, documentações relevantes para as APIs, exemplos de códigos fonte e tutoriais para o sistema operacional Android. O SDK também permite que sejam criados bancos de dados *SQLite* para os aplicativos. O SQLite é um banco de dados leve, de código fonte aberto e que possui as características básicas, tais como tabelas, gatilhos e visões, necessárias para a estruturação de dados [Lellis e Cruz 2015]. Estes bancos de dados são armazenados normalmente no subdiretório */data/data/<app>/databases*. No Capítulo 4, estes bancos de dados que contêm importante informações do usuário serão analisados.

O ADB, citado anteriormente como parte do SDK, é o Android Debug Bridge que permite a comunicação entre um computador e um dispositivo Android. É um programa cliente servidor, que inclui três componentes: um cliente, que envia comandos e é executado na sua máquina de desenvolvimento; um serviço (*daemon*) que faz os comandos em um dispositivo e é realizado como um processo de *background* e um servidor que gerencia a comunicação entre o cliente e o *daemon*. Entre as várias utilidades desta ferramenta, destacam-se: a instalação de aplicativos, execução de comandos diretamente no *shell* do dispositivo e cópia de arquivos entre o computador e o *smartphone* e vice-versa [Lellis e Cruz 2015]. Para que seja possível o uso do ADB com um dispositivo conectado via cabo USB, a função USB *debugging* (depuração USB) tem de estar habilitada nas configurações de sistema do dispositivo, esta pode ser habilitada quando as opções de desenvolvedor são ativadas. Se o dispositivo possuir a tela bloqueada, o perito tem que, primeiramente, desbloqueá-la para então acessa o menu de configurações e ativar a depuração USB.

Tabela 2.2: Importante Partições do Android [Wilson]

Partição	Explicação
/boot	Inclui o *kernel* do Android
/cache	Cache de aplicativos
/data	Armazena dados do usuário
/data/data	Onde os aplicativos e seus dados são armazenados
/dev	Informações sobre o dispositivo
/mnt/asec	Aplicativos criptografados
/mnt/emmc	Cartão SD interno
/mnt/sdcard	Cartão SD externo
/proc	Informações sobre processos
/recovery	É utilizado no modo de recuperação
/system	ROM do sistema (somente leitura)

Existem partições (divisão lógica de um dispositivo de armazenamento de dados) típicas nos *smartphones* com Android, estas são: *data* (dados do usuário), *boot*, *cache*, *recovery*. A maioria dos dados de um usuário estarão armazenados na partição */data*, mas informações importantes também podem ser encontradas no cartão SD. A Tabela 2.2 mostra as principais partições informadas anteriormente e explica o que cada uma delas armazena.

Capítulo 3

Métodos Propostos e Ferramentas Utilizadas

A partir do problema apresentado no Capítulo 1, serão descritas soluções que visam facilitar a análise forense de dispositivos com sistema operacional Android. Foram consideradas ferramentas com suporte ao sistema operacional Linux, para que, ao final do projeto, ocorra a criação de um *Live CD* com todos os roteiros e análises realizadas. O objetivo também foi utilizar ferramentas *open source* que tenham recursos equivalentes à ferramentas com foco comercial.

3.1 Descrição dos Métodos

Para realizar a análise dos dispositivos, serão considerados três métodos. O primeiro é a utilização de um módulo que será incorporado, por meio de uma compilação, ao *kernel* do *smartphone*. O outro método será o de instalar um aplicativo no telefone com sistema operacional Android, que capturará os dados do usuário (agenda de contatos, registros de chamadas, SMS's). A última técnica utilizada, é a criação de uma imagem de partições do dispositivo.

Para se chegar nos métodos descritos, foram utilizadas um conjunto de recursos detalhados conforme algumas ferramentas que serão descritas nas próximas seções. As primeiras utilizadas foram o LiME e o *Volatility Framework*. As duas se complementam, pois usa-se o LiME para realizar a extração da memória volátil de dispositivos e o *Volatility Framework* analisa estas memórias coletadas através do uso de plugins. Depois utilizou-se as ferramentas *The Sleuth Kit* - TSK e *Autopsy* que também se complementam, pois o *Autopsy* é uma interface gráfica para se utilizar o TSK. Como complemento, também foram analisadas as ferramentas *Scalpel*, *Foremost* e *AFLogical OSE*. Os meios de instalação das ferramentas e também de outros *softwares* que são pré-requisitos para o trabalho, estão descritos no Anexo I.

3.2 LiME - Linux Memory Extractor

O LiME é um *Loadable kernel Module* - LKM, isto é, um módulo carregável do *kernel*. Ele permite que seja feita a aquisição da memória volátil de dispositivos que tenham o Linux como sistema operacional e também os que são baseados em Linux, como é o caso do Android. É a primeira ferramenta capaz de capturar a memória volátil de um dispositivo Android. Outra característica importante é que consegue minimizar a interação entre os processos de usuário e de *kernel*, o que permite que a memória capturada seja de âmbito mais forense do que outras ferramentas designadas para a aquisição de memória de sistemas Linux. A obtenção dos dados coletados pode ser feita através da interface de rede ou diretamente para o cartão de memória do dispositivo [LiME].

No Linux o arquivo */proc/iomem* mostra o estado presente da memória do sistema para cada dispositivo físico. LiME basicamente copia estes segmentos de RAM que estão listados em */proc/iomem* que são chamados de RAM do sistema. O usuário pode escolher entre três formatos de arquivo diferentes para gravar o *dump* da memória [LiME].

- ***Raw***: toda RAM do sistema é simplesmente concatenada;
- ***Padded***: começando do endereço físico 0, tudo o que não pertencer à RAM do sistema é completado com zeros;
- ***Lime***: funciona como o formato *raw*, mas os segmentos de memória são precedidos com um cabeçalho de 32 bytes contendo informações de espaço de endereço.

Trabalhou-se com o formato *lime* durante as análises. Esta escolha se deve principalmente por causa da outra ferramenta a ser trabalhada, o *Volatility*, que contém um espaço de endereço em que se define o uso do formato *lime*. No caso de se extrair a memória de um dispositivo Android, existe uma série de pré-requisitos para que tudo ocorra de forma eficiente. Primeiramente, deve-se instalar o Android Studio e, com ele, uma ferramenta extra, o NDK, ambas instalações estão detalhadas no Anexo I. Com estas duas ferramentas, é possível fazer o *cross-compile* do *kernel* (que deve ser previamente compilado) do dispositivo com o LiME.

Quando o investigador for utilizar o módulo do LiME, já pronto, carregando-o ao dispositivo, o mesmo deve informar um diretório para onde o *dump* da memória seria copiado ou uma porta TCP para que o dispositivo se conecte diretamente por ela com o auxílio do *netcat*. Como dito, o módulo do LiME foi implementado para que houvesse o mínimo de interação possível entre os processos do usuário e o *kernel*, por isso, o *dump* da memória é escrito diretamente do *kernel*. Isto economiza um grande número de chamadas do sistema e outras atividades do *kernel* que outras ferramentas utilizam, como, por exemplo, a necessidade de chamadas de leitura e escrita para cada bloco de dados requisitados pela memória do dispositivo. O módulo também tenta evitar o uso dos *buffers* do sistema de arquivo do *kernel* e também *buffers* de rede para que seja minimizada ainda mais a contaminação da memória volátil durante o processo de aquisição.

LiME utiliza o comando *insmod* para carregar o módulo no dispositivo, juntamente com os argumentos requeridos para sua execução. Deve-se seguir o formato:

```
insmod ./lime.ko "<path=<outfile|tcp:<port>> format=<raw|padded|lime> [dio=<0|1>]"
```

Onde [LiME]:

- ***path*** (obrigatório): *outfile* é o caminho onde o *dump* será salvo, deve-se incluir também o nome do arquivo a ser escrito no sistema local e *tcp:port* é a porta de rede para se comunicar se a transferência for realizada com o auxílio do *netcat*;
- ***format*** (obrigatório): escolha de um dos formatos possíveis explicados anteriormente;
- ***dio*** (opcional): se o valor determinado for 1, se comporta como a tentativa de se habilitar o *Direct IO* e se a opção for 0 (esta é a opção padrão) não se tenta habilitar o *Direct IO*;
- ***localhostonly*** (opcional): se a opção for determinada como 1, restringe a conexão TCP apenas para o *localhost*, enquanto a opção 0 é a padrão, em que a conexão pode ser efetuada em qualquer porta;
- ***timeout*** (opcional): se o processo de ler/escrever uma página da memória e demorar mais do que o *timeout* especificado (em milissegundos), então, assume-se que o intervalo é ruim e o mesmo é pulado. Para desabilitar esta configuração, o *timeout* pode ser especificado como 0. A configuração padrão é 1000 (1 segundo).

Como foi citado, existem dois modos de aquisição da memória [Sylve et al. 2012]:

- **Por uma conexão TCP**: utiliza-se o ADB para configurar o encaminhamento de porta, a partir de uma porta TCP específica no dispositivo para uma outra no *localhost*. O uso do ADB para transferência de rede, elimina a necessidade de modificar a configuração de rede do dispositivo ou introduzir um par de rede sem fio, em que os dados são transferidos via USB. Depois da configuração inicial, mais duas etapas são feitas para a aquisição da memória. A primeira é aquela em que o dispositivo alvo tem de estabelecer uma conexão na porta TCP especificada e então deve conectar-se ao dispositivo, a partir do computador hospedeiro, com o auxílio do *netcat*. Quando o *socket* se conecta, o módulo do *kernel* automaticamente envia a imagem RAM coletada para o computador do investigador. A Figura 3.1 mostra os comandos necessários para se estabelecer a conexão e para ativar o módulo no dispositivo. Já a Figura 3.2 mostra que, para completar a conexão, é preciso estabelecer no computador do investigador uma conexão à mesma porta configurada no dispositivo via *netcat* e então direcionar a saída de informações a um arquivo.
- **Diretamente para o cartão de memória do dispositivo**: em alguns casos, como quando o investigador quer ter certeza de que nenhum *buffer* de rede seja sobre-escrito, as aquisições baseadas em disco, podem ser melhores do que as aquisições feitas pela rede. Por este motivo, o LiME provê a opção de escrever imagens de memórias diretamente para o sistema

```
raquel@raquel:~$ adb push lime.ko /sdcard/lime.ko
152 KB/s (6608 bytes in 0.042s)
raquel@raquel:~$ adb forward tcp:4444 tcp:4444
raquel@raquel:~$ adb shell
# insmod /sdcard/lime.ko "path=tcp:4444 format=lime"
```

Figura 3.1: Estabelecimento da conexão TCP entre o dispositivo e o *host*

```
raquel@raquel:~$ nc localhost 4444 > ram.lime
```

Figura 3.2: *Netcat* para se obter a saída da memória a ser coletada diretamente para o *host*

de arquivos do dispositivo alvo. No Android, o lugar mais lógico para esta escrita é o cartão de memória do dispositivo. Como, o cartão de memória pode, potencialmente, conter evidências importantes, o investigador que escolher esta alternativa pode gerar uma imagem do cartão de memória antes de se fazer qualquer outra coisa. Mesmo que este processo viole a típica *ordem de volatilidade* – regra de ouro na aquisição forense –, ou seja, a obtenção da informação mais volátil em primeiro lugar, é necessário preservar adequadamente as provas. Da mesma maneira que o modo de conexão por TCP, a primeira coisa a ser feita é copiar o módulo do LiME para o dispositivo e depois já usa-se o comando *insmod* para que o *dump* da memória seja feito e salvo no cartão de memória. A Figura 3.3 demonstra os passos a serem feitos.

```
raquel@raquel:~$ adb push lime.ko /sdcard/lime.ko
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
137 KB/s (6608 bytes in 0.046s)
raquel@raquel:~$ adb shell
# insmod /sdcard/lime.ko "path=/sdcard/ram.lime format=lime"
```

Figura 3.3: Comandos para se obter a memória e salvá-la diretamente no cartão de memória do dispositivo

Assim que o processo de aquisição estiver finalizado, o dispositivo pode ser desligado para que o cartão de memória seja removido e, então, pode-se transferir a memória para a máquina onde está sendo feita a investigação. Se o dispositivo não puder ser desligado, o ADB pode ser usado para fazer a transferência de informação, através da função *pull*.

3.3 *Volatility Framework*

Após a captura da memória volátil, é preciso uma ferramenta que auxilie na análise dos dados e esta será o *Volatility*. O *Volatility Framework* é uma coleção de ferramentas, implementada em *Python* para a extração de artefatos digitais de amostras de memórias voláteis (RAM). As técnicas de extração são realizadas completamente independentes do sistema que está sendo investigado. O *Volatility* suporta a investigação da memória de vários sistemas operacionais, como o Windows, Linux, Mac OSX e, consequentemente, com o suporte para o Linux assim será para o Android. Para que seja possível a análise da memória RAM de um dispositivo com sistema operacional Android, têm que ser considerados os pré-requisitos discutidos anteriormente. Um destes, para o funcionamento correto desta ferramenta, é a instalação do *dwarfdump* que está descrita no Anexo I.

Um dos aspectos mais importantes desta ferramenta é a criação de perfis. Para cada combinação de uma versão específica de um SO e uma arquitetura de hardware, é necessário que sejam criados perfis separados para cada uma. Cada perfil inclui as seguintes particularidades [Ligh et al. 2014]:

- Um conjunto de *Vtypes*, sobreposições e classes de objetos;
- Metadados, entre eles o nome do SO, a versão do *kernel* e o número de compilação;
- Índices e nomes de chamadas do sistema;
- Variáveis globais que podem ser encontradas em endereços codificados em alguns sistemas operacionais;
- Tipagem de baixo nível para linguagens nativas (geralmente C), incluindo o tamanho para inteiros, inteiros longos e assim por diante;
- *System map*: endereços de variáveis globais e funções críticas.

Criar um perfil no *Volatility* significa gerar um conjunto de *Vtypes* (definições de estrutura) e um arquivo *System.map* para uma versão particular do *kernel* e reuni-las em um arquivo *.zip*. Os *Vtypes* podem ser extraídos com o auxílio da ferramenta *dwarfdump* que analisa as informações de *debbuging* de arquivos ELF, como o *kernel* do Linux e módulos do *kernel*, entregando ao *Volatility* exatamente o que ele precisa. O arquivo *System.map* é necessário porque nele está contido o nome e endereço de cada estrutura de dados estática utilizada no *kernel*.

Depois da instalação e criação do perfil do dispositivo a ser analisado, pode-se examinar a memória RAM. Como o Android é baseado em Linux, os comandos a serem utilizados para as análises são aqueles atribuídos ao Linux, nas referências da ferramenta [Volatility]. Na Figura 3.4 há a descrição do uso padrão dos *plugins* do *Volatility*. O parâmetro *memory capture path* deve ser o caminho completo de onde foi salva a memória RAM coletada do dispositivo e, logo depois, usa-se o nome do perfil criado anteriormente (*–profile=android*). Alguns *plugins* apresentam opções específicas que podem ser incluídas na requisição da análise. Para descobrir as opções de *plugins* disponíveis para se usar a partir de um certo perfil criado, pode-se usar o parâmetro *-h*.

```
python volatility.py -f [memory capture path] --profile=android [plugin op-
tions] [plugin name]
```

Figura 3.4: Uso do *Volatility* em uma captura de memória de um dispositivo Android

Nem todos os *plugins* criados para uma análise de SO Linux podem ser utilizados com uma análise de um dispositivo cujo SO é o Android. Os *plugins* e suas respectivas funções que funcionam com um perfil de um dispositivo com sistema operacional Android são [Volatility]:

- **linux_arp**: retorna a tabela ARP do dispositivo;
- **linux_check_afinfo**: verifica os ponteiros de funções de operação de protocolos de redes;
- **linux_dentry_cache**: reúne arquivos do *dentry cache*, que são os objetos de entrada de diretório que são armazenados;
- **linux_dmesg**: reúne o *buffer* de mensagens do sistema;
- **linux_dump_map**: escreve determinados mapeamentos de memória para o disco;
- **linux_find_file**: recupera sistemas de arquivos da memória;
- **linux_ifconfig**: reúne e retorna interfaces de rede ativas no dispositivo;
- **linux_iomem**: é equivalente à saída do arquivo /proc/iomem do Linux, ou seja, retorna os endereços de memória para os dispositivos de entrada e saída;
- **linux_lsmod**: retorna informações sobre os módulos carregáveis do *kernel*;
- **linux_lsof**: lista arquivos abertos;
- **linux_memmap**: faz o *dump* do mapeamento da memória para tarefas do linux;
- **linux_mount**: retorna os dispositivos e sistemas de arquivos que foram montados no sistema;
- **linux_mount_cache**: retorna os dispositivos e sistemas de arquivos que foram montados no sistema provenientes do *cache* da memória do *kernel* (kmem_cache);
- **linux_pidhashtable**: enumera processos através da *hash table* dos PID's;
- **linux_pkt_queues**: escreve filas de pacotes por processo para o disco;
- **linux_proc_maps**: reúne e retorna mapas de processos para o Linux;
- **linux_psaux**: reúne e retorna processos juntamente com os comandos executados. Também retorna a hora em que o processo foi iniciado;
- **linux_pslist**: reúne e retorna tarefas ativas;

- **linux_pslist_cache**: reúne e retorna tarefas provenientes do *cache* da memória do *kernel* (kmem_cache);
- **linux_pstree**: apresenta os processos em forma de árvore, desde sua raiz (começando pelo processo *init*) até os diretórios filhos da raiz;
- **linux_psxview**: encontra processos ocultos dentre várias listas de processos;
- **linux_route_cache**: recupera o *cache* de roteamento da memória;
- **linux_sk_buff_cache**: recupera pacotes do *sk_buff*, que são *buffers* onde estão depositadas informações referentes aos pacotes de rede relacionados a memória do *kernel*;
- **linux_slabinfo**: retorna o mesmo resultado que do arquivo */proc/slabinfo* do Linux, o *slabinfo* mantém as estatísticas sobre os objetos armazenados na memória;
- **linux_tmpfs**: recupera sistemas de arquivos *tmpfs* da memória;
- **linux_vma_cache**: reúne cada *Virtual Memory Area* - VMA armazenada em *cache*.

O uso dos principais *plugins* serão detalhados no Capítulo 4.

3.4 *The Sleuth Kit* (TSK)

Trata-se de uma coleção de ferramentas de linha de comando, isto é, uma biblioteca em C e ferramentas de análise forense de sistemas de volume, que permitem a análise de imagens de discos e a recuperação de arquivos provenientes das mesmas imagens. Esta é utilizada em conjunto com o *Autopsy* e outras ferramentas, atuando no *background* das operações [SleuthKit].

A ferramenta de sistema de volumes (gerenciamento de mídia) permite que seja feita a avaliação do *layout* dos discos e outras mídias. O TSK suporta partições DOS, BSD, Mac, *sun slices* e discos GPT. Com estas ferramentas, pode-se identificar onde as partições estão localizadas e então ser extraídas para que sejam examinadas com as ferramentas adequadas.

Uma análise completa também requer mais do que somente a análise de sistemas de arquivos e volume. Contudo, somente uma ferramenta pode não conseguir dar suporte para todos os tipos de arquivos e técnicas de análise. O TSK *Framework* permite que sejam incorporados facilmente módulos de análise de arquivos que foram escritos por outros desenvolvedores [SleuthKit].

Quando uma análise completa de um sistema é feita, a utilização de ferramentas baseadas em linhas de comando pode tornar-se tediosa e complicada. Por isso, o TSK possui uma interface gráfica chamada *Autopsy* para suas ferramentas, permitindo que a condução de investigações seja feita de forma mais fácil. O *Autopsy* provê gerenciamento de casos, integridade da imagem, procura por palavras-chave, e outras operações que serão descritas na próxima seção.

Quanto à entrada de dados, o TSK analisa sistemas de arquivos e imagens de disco nos formatos: *raw* (*dd*), *Expert Witness* (*EnCase*) e *AFF*. Também tem suporte para os formatos: NTFS, FAT,

ExFAT, UFS 1, UFS 2, EXT2FS, EXT3FS, EXT4, HFS e YAFFS2. O TSK utiliza algumas técnicas de pesquisa específicas [SleuthKit]:

- Lista nomes de arquivos ASCII e *Unicode* alocados e excluídos;
- Mostra os detalhes e conteúdos de todos os atributos de arquivos NTFS;
- Mostra detalhes de sistemas de arquivos e estruturas de metadados;
- Cria linhas do tempo das atividades dos arquivos, que podem ser importadas para a posterior criação de gráficos e relatórios;
- Procura arquivos *hash* em um banco de dados de *hashes*;
- Organiza arquivos baseando-se em seus tipos (por exemplo, todos executáveis, *jpegs*, e documentos são separados). Páginas de *thumbnails* podem ser feitas de imagens gráficas para rápidas análises.

3.5 *Autopsy*

A ferramenta *Autopsy* é uma plataforma forense digital com interface gráfica, que é executada com o TSK e outras ferramentas. Tanto o *Autopsy* como o TSK são de código aberto e podem ser executados em plataformas *UNIX*. Como o *Autopsy* é baseado em HTML, é possível conectar ao servidor da ferramenta de qualquer plataforma, usando um navegador HTML. Provê uma interface de gerenciador de arquivos e mostra detalhes sobre dados deletados e estruturas de sistemas de arquivos. Permite que seja feita uma análise eficiente de *hard drives* e *smartphones* [Autopsy].

Possui dois modos de análises: uma *dead analysis* e uma *live analysis* [Autopsy]:

- ***Dead analysis***: ocorre quando um sistema de análise todo *sui generis* é usado para examinar dados de um sistema suspeito. Neste caso, *Autopsy* e TSK são executados em um ambiente confiável, geralmente em um laboratório. Suporta-se os seguintes formatos de arquivos: *raw*, *Expert Witness - EnCase* e *AFF*.
- ***Live analysis***: ocorre quando o sistema suspeito está sendo analisado, enquanto o mesmo está sendo executado. Neste caso, as ferramentas são executadas a partir de um CD em um ambiente não confiável. Este modo é, frequentemente, utilizado durante uma resposta a um incidente, enquanto o mesmo está sendo confirmado. Depois que este é confirmado, pode-se adquirir a imagem do sistema e, então, a *dead analysis* é realizada.

Existem várias técnicas de procura de evidências utilizadas pelo *Autopsy* [Autopsy]:

- **Listagem de arquivos**: são analisados os arquivos e diretórios, incluindo o nome de arquivos deletados e aqueles que têm nomes baseados em *Unicode*;

- **Conteúdo dos arquivos**: o conteúdo pode ser visualizado nos formatos *raw*, *hex* e aqueles que estão em ASCII, podem ser extraídos. Quando os dados são interpretados, o *Autopsy* faz umma limpeza para previnir danos no sistema local analisado;

- **Banco de dados de *hashes***: faz a busca de arquivos desconhecidos em um banco de dados de *hashes* para identificar rapidamente se são maliciosos ou não. O *Autopsy* utiliza o NSRL da NIST e também banco de dados criados por usuários;

- **Classificação de tipos de arquivos**: classifica os arquivos baseando-se nas assinaturas internas dos mesmos, para identificar aqueles de tipos conhecidos. O *Autopsy* também pode extrair somente imagens gráficas (incluindo *thumbnails*). A extensão do arquivo também será comparada ao formato para identificar os que teriam suas extensões modificadas para escondê-lo;

- **Linha do tempo da atividade de um arquivo**: em alguns casos, ter uma linha do tempo da atividade de um arquivo pode ajudar na identificação de áreas que contenham evidências importantes. O *Autopsy* pode criar linhas do tempo que contenham entradas para os momentos em que os arquivos foram acessados ou modificados, sendo estes alocados ou não;

- **Procura por palavras-chave**: procuras de palavras-chave podem ser feitas usando elementos ASCII e o *grep* de expressões regulares. As procuras podem ser realizadas tanto em uma imagem de um sistema de arquivos em geral, como num espaço não alocado. Expressões, palavras que são frequentemente buscadas podem ser facilmente configuradas para pesquisas automáticas;

- **Análises de metadados**: estruturas de metadados contêm detalhes sobre arquivos e diretórios. O *Autopsy* permite que sejam visualizados os detalhes de qualquer estrutura de metadados em um sistema de arquivos. Isto é eficiente para a recuperação de conteúdos deletados. Deve ser feita uma pesquisa nos diretórios para identificar o caminho completo do arquivo que tem a estrutura alocada;

- **Análise da unidade de dados**: unidades de dados são onde os conteúdos dos arquivos são armazenados. O *Autopsy* permite que seja visualizado o conteúdo de qualquer unidade de dados em uma variedade de formatos, incluindo ASCII, *hexdump* e *strings*. O tipo do arquivo também é informado e a ferramenta procurará as estruturas de metadados para identificar em qual lugar está alocada a unidade de dados;

- **Detalhes da imagem do sistema**: este modo provê informações que são úteis durante a recuperação de dados. Descreve informações acerca da imagem feita do sistema.

Além das várias técnicas de procura de evidências descritas acima, o *Autopsy* também provê algumas opções de gerenciamento dos casos de investigação [Autopsy]:

- **Gerenciamento de casos**: as investigações são organizadas por casos, que podem conter um ou mais *hosts*. Cada *host* é configurado para ter seu próprio fuso horário, para que os

momentos de acesso e modificação mostrados sejam os mesmos que o usuário do dispositivo analisado tenha utilizado. Cada *host* pode conter uma ou mais imagens de sistemas de arquivos para serem analisadas;

- **Sequenciador de eventos**: eventos baseados no tempo podem ser adicionados a partir de atividades de arquivos ou IDS e *logs de firewall*. O *Autopsy* organiza os eventos para que a sequência de incidentes possam ser mais facilmente determinada;
- **Notas/Observações**: podem ser salvas por cada *host* ou por cada investigação. Isto permite que rápidas anotações possam ser feitas sobre arquivos e estruturas. A localização original pode ser facilmente recuperada com apenas um clique, quando as notas forem posteriormente revisadas. Todas são armazenadas em um arquivo ASCII;
- **Integridade da imagem**: é importante certificar-se de que os arquivos não sejam modificados durante a análise. Por padrão, o *Autopsy* gera um valor MD5 para todos os arquivos que são importados ou criados. A integridade dos arquivos que sejam usados, pode ser validada a qualquer momento;
- **Relatórios**: podem ser criados relatórios em formato ASCII para arquivos e outras estruturas de sistemas de arquivos;
- ***Logging***: registros para auditoria são criados em cada caso, *host* e nível de investigação, para que certas ações possam ser recuperadas;
- ***Open Design***: o código do *Autopsy* é sempre aberto e os arquivos que o mesmo usa estão em formato *raw*. Todos os arquivos de configurações estão em texto ASCII e os casos são organizados por diretórios. Isto facilita a exportação de dados e sua arquivação;
- **Modelo cliente-servidor**: como o *Autopsy* é baseado em HTML, não é preciso que o investigador esteja no mesmo sistema que a imagem dos arquivos. Isto permite que múltiplas investigações utilizem o mesmo servidor e que se conectem a seus próprios sistemas.

3.6 Captura de imagem com o *dd*

É preciso que seja feita uma imagem das partições a serem analisadas do dispositivo em um dos formatos aceitos pelo *Autopsy*. A ferramenta utilizada para esta captura será o *dd*, pois, consegue gerar resultados com formatos de imagem *raw*, ou seja, realiza uma cópia *bit* a *bit* dos dados *raw* de um disco, partição ou volume, sem adições de outros tipos de dados ou exclusões. Como o *dd* foi implementado para sistemas baseados em Linux, pode-se utilizá-lo também para plataformas Android [Tamma e Tindall 2015]. A imagem pode ser salva com as seguintes nominações:

- *dd*
- *dmg*
- *img*

- *raw*

A imagem é extraída através de apenas uma linha de comando que ainda pode utilizar alguns parâmetros específicos para aprimorar as capturas. O exemplo apresentado na Figura 3.5 demonstra a aquisição da imagem da partição */system* (/dev/mtd/mtd0) de um dispositivo Android.

```
root@generic:/ # dd if=/dev/mtd/mtd0 of=/sdcard/teste.dd bs=4096
```

Figura 3.5: Exemplo de uso da ferramenta *dd* em dispositivo Android

Existem vários parâmetros que podem ser utilizados com a ferramenta. A seguir, os mais comuns em uma análise forense [Tamma e Tindall 2015]:

- ***if*** (uso obrigatório): especifica o caminho do arquivo de entrada do qual será extraída a imagem;
- ***of*** (uso obrigatório): especifica o caminho do arquivo de saída em que será escrita a imagem;
- ***bs*** (uso opcional): especifica o tamanho do bloco e os dados são lidos e escritos no tamanho especificado. O padrão é 512 *bytes* se nada for especificado;
- ***conv*** (uso opcional): especifica opções de conversão, que é utilizada com alguns atributos:
  - ***noerror***: a captura não para, se um erro de leitura for encontrado;
  - ***notrunc***: com esta opção, o arquivo de saída não é truncado;
  - ***sync***: se ocorrer um erro, o bloco em que este se encontra será preenchido com espaços nulos; usualmente, o *sync* é utilizado juntamente com o parâmetro *noerror*.

É preciso ter muito cuidado para que não se troque os parâmetros *if* e *of*, pois isto pode resultar na sobrescrição dos diretórios alvos. Quando se trabalha com o *dd* em dispositivos Android, é importante estar ciente de que o lugar em que o *dump* da memória será armazenado tem de ser um lugar seguro. Na maioria das vezes, as informações são transferidas para o cartão de memória, de preferência, este pode ser um cartão novo do investigador [Tamma e Tindall 2015].

O problema nesta situação de se transferir para um cartão de memória, é que alguns dispositivos não permitem sua extração ou até mesmo não têm *slot* para um cartão, então a maneira mais segura de se transferir os dados seria, pela rede, por uma conexão TCP, com o auxílio da ferramenta *netcat*.

3.7 *Scalpel*

Além do uso do *Autopsy*, outras ferramentas e técnicas podem ser utilizadas para analisar imagens em formato *raw*. O método de *carving* é muito útil em análises forenses, porque permite que dados escondidos ou deletados sejam recuperados. É um processo de remontagem de arquivos de fragmentos, na ausência de metadados de sistema de arquivos. O *carving* de arquivos recupera

aqueles de espaços não alocados em um disco ou partição baseados meramente na estrutura do arquivo e do seu conteúdo [Tamma e Tindall 2015].

Uma das ferramentas que conseguem fazer este tipo de técnica é o *Scalpel*. É uma ferramenta de código aberto que analisa o bloco do armazenamento do banco de dados e identifica os arquivos deletados e recupera os mesmos. A instalação da ferramenta está descrita no Anexo I. No diretório onde a ferramenta é instalada, */etc/Scalpel/*, há o arquivo *Scalpel.conf* que contém informações sobre os tipos que são suportados. O *Scalpel.conf* precisa ser modificado para que sejam incluídos tipos que tenham relação com o Android. Utilizou-se uma amostra presente no endereço eletrônico descrito na referência [Buchanan], para a utilização da ferramenta. Também pode-se utilizar o arquivo, que é obtido a partir da instalação e somente acrescentar os formatos excedentes ou comentar as linhas onde são descritos outros formatos que não serão úteis para a investigação. A Figura 3.6 mostra um caso geral de uso da ferramenta.

```
raquel@raquel:~$ scalpel -c /etc/scalpel/scalpel.conf /var/lib/autopsy/Android6_
Analise/host1_android/images/data.dd -o /home/raquel/0/scalpel/
Scalpel version 1.60
Written by Golden G. Richard III, based on Foremost 0.69.

Opening target "/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd"

Image file pass 1/2.
/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd:   0.5%    10.0 M
/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd:   1.0%    20.0 M
/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd:   1.5%    30.0 M
/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd:   2.0%    40.0 M
/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd:   2.4%    50.0 M
/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd:   2.9%    60.0 M
/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd:   3.4%    70.0 M
```

Figura 3.6: Uso da ferramenta *Scalpel*

Como visto nesta mesma figura, só é preciso utilizar: o caminho onde o arquivo *Scalpel.conf* estiver armazenado, o caminho da imagem a ser analisada e onde as informações coletadas serão armazenadas. Logo depois, a ferramenta analisará a imagem e retornará ao que está descrito na Figura 3.7, onde são identificados os tipos de arquivos que foram recuperados e também a quantidade dos mesmos.

```
/var/lib/autopsy/Android6_Analise/host1_android/images/data.dd: 100.0%     2.0 G
B    00:00 ETAAllocating work queues...
Work queues allocation complete. Building carve lists...
Carve lists built.  Workload:
jpg with header "\xff\xd8\xff\xe0\x00\x10" and footer "\xff\xd9" --> 55 files
gif with header "\x47\x49\x46\x38\x37\x61" and footer "\x00\x3b" --> 1 files
gif with header "\x47\x49\x46\x38\x39\x61" and footer "\x00\x00\x3b" --> 5 files
jpg with header "\xff\xd8\xff\xe0\x00\x10" and footer "\xff\xd9" --> 94 files
jpg with header "\xff\xd8\xff\xe1" and footer "\xff\xd9" --> 2 files
png with header "\x50\x4e\x47\x3f" and footer "\xff\xfc\xfd\xfe" --> 28 files
bmp with header "\x42\x4d\x3f\x3f\x00\x00\x00" and footer "" --> 40 files
tif with header "\x49\x49\x2a\x00" and footer "" --> 8 files
tif with header "\x4d\x4d\x00\x2a" and footer "" --> 26 files
Carving files from image.
Image file pass 2/2.
```

Figura 3.7: Arquivos que foram recuperados pela ferramenta *Scalpel*

Na pasta onde foi armazenada a saída com as informações coletadas, os resultados são divididos de acordo com seu formato de arquivo e também apresenta-se um registro com a descrição dos

resultados. Estas informações são descritas na Figura 3.8. Alguns arquivos foram completamente recuperados, enquanto outros não puderam ser recuperados.

Figura 3.8: Pasta em que foram salvas informações são divididas de acordo com o formato dos arquivos recuperados

3.8 *Foremost*

A ferramenta *Foremost* utiliza a mesma técnica de *carving* de arquivos que o *Scalpel*. Após sua instalação (descrita no Anexo I), no diretório */etc/* há o arquivo de configuração *Foremost.conf*, que, assim como o arquivo do *Scalpel*, pode ser modificado de acordo com formatos específicos que o investigador queira recuperar. Depois das configurações serem finalizadas, pode-se, através do comando *Foremost -h*, descobrir os parâmetros que podem ser utilizados na análise das imagens. A Figura 3.9 mostra o resultado desta operação.

```
raquel@raquel:~$ foremost -h
foremost version 1.5.7 by Jesse Kornblum, Kris Kendall, and Nick Mikus.
$ foremost [-v|-V|-h|-T|-Q|-q|-a|-w-d] [-t <type>] [-s <blocks>] [-k <size>]
        [-b <size>] [-c <file>] [-o <dir>] [-i <file]

-V  - display copyright information and exit
-t  - specify file type.  (-t jpeg,pdf ...)
-d  - turn on indirect block detection (for UNIX file-systems)
-i  - specify input file (default is stdin)
-a  - Write all headers, perform no error detection (corrupted files)
-w  - Only write the audit file, do not write any detected files to the disk
-o  - set output directory (defaults to output)
-c  - set configuration file to use (defaults to foremost.conf)
-q  - enables quick mode. Search are performed on 512 byte boundaries.
-Q  - enables quiet mode. Suppress output messages.
-v  - verbose mode. Logs all messages to screen
```

Figura 3.9: Comando para identificar os parâmetros que podem ser utilizados pela ferramenta *Foremost*

Para um teste de um caso geral, não utilizou-se o arquivo de configuração. Assim, a ferramenta procurou por todos os formatos possíveis, sem especificações. A Figura 3.10 mostra como a ferramenta é utilizada. Só é preciso que sejam informados os locais da imagem a ser analisada e onde serão salvas as informações. Os dados encontrados foram armazenados no diretório */home/raquel/0/Foremost/* e assim como no *Scalpel*, os resultados são salvos separados por pastas de acordo com os formatos de arquivos.

Assim como no *Scalpel*, um relatório com os registros dos arquivos encontrados também é gerado, com a informação de quantos arquivos foram recuperados, na Figura 3.11 há um exemplo

```
raquel@raquel:~$ foremost /var/lib/autopsy/Android6_Analise/host1_android/image
s/data.dd -o /home/raquel/0/foremost/
Processing: /var/lib/autopsy/Android6_Analise/host1_android/images/data.dd
```

Figura 3.10: Instrução de como se utiliza a ferramenta

de como isto é informado.

```
Finish: Mon Nov 21 16:09:29 2016

4251 FILES EXTRACTED

jpg:= 54
gif:= 5
bmp:= 1
mov:= 1
htm:= 17
zip:= 49
png:= 4124
------------------------------------------------

Foremost finished at Mon Nov 21 16:09:29 2016
```

Figura 3.11: Quantidade de arquivos recuperados pela ferramenta

3.9 *AFLogical OSE*

Esta ferramenta provê um *framework* básico para a extração de dados de dispositivos Android, usando provedores de contéudo e salvando os dados para o cartão de memória do dispositivo [NowSecure]. As informações que consegue capturar incluem:

- Contatos;
- *Logs* das Chamadas;
- SMS's;
- MMS's;
- Partes de MMS's;
- Informações sobre o dispositivo e seus aplicativos.

Para utilizá-lo, primeiramente, deve-se fazer o *download* e então instalar o aplicativo *.apk* no dispositivo em que se deseja obter as informações. O *download* pode ser feito no endereço eletrônico, citado na referência [NowSecure]. Para o segundo passo de instalação no telefone, este tem de estar em modo *root*, pois, assim, o ADB pode ser utilizado para instalar a ferramenta.

As informações são salvas em uma pasta chamada *forensics* criada pelo próprio aplicativo, no dispositivo que está sendo investigado e dentro desta se encontram as informações que são escolhidas anteriormente pelo investigador. Para analisar estas informações, pode-se utilizar uma das funções do ADB, que é a de *pull*, que obtém informações do dispositivo para o computador utilizado.

Ao acessar o diretório em questão, são encontradas as informações que foram pedidas pelo usuário em arquivos *.csv*, em formato de tabelas e também um arquivo *.xml*, onde estão contidas informações gerais sobre o dispositivo, como seu International Mobile Equipment Identity - IMEI, versão, marca e descrição de todos os aplicativos presentes no mesmo.

Capítulo 4

Análises e Resultados

Este Capítulo trata das análises e dos seus respectivos resultados, a partir das ferramentas descritas. Serão considerados três cenários para os exames de memória volátil e persistente.

4.1 Cenários Analisados

Como foi justificado no Capítulo 1, foram utilizados três cenários para que houvesse uma ampla cobertura amostral do cenário real. O primeiro trata de um emulador criado com o Android Studio, de um dispositivo Nexus 5, com a versão 4.0.3 - *Ice Cream Sandwich* do Android. O segundo cenário foi também um emulador de um dispositivo Nexus 5, porém, este utilizou-se da versão 6.0 - *Marshmallow* do Android. O último cenário foi um estudo de caso real, de um dispositivo Samsumg, modelo Galaxy S4, GT - I9515L e versão 5.0.1 - *Lollipop* do Android, este foi analisado em três sub-casos diferentes.

A seção 4.2 tratou do primeiro cenário descrito. Analisou-se a memória volátil do dispositivo, com as ferramentas LiME e *Volatility Framework* e também utilizou-se a *AFLogical OSE*. Esperou-se coletar informações sobre os aplicativos instalados no dispositivo e seus mecanismos de segurança quanto às informações sensíveis do usuário. Também foram coletados dados úteis a uma investigação – demonstrado no Capítulo 2 –, como registros de chamadas, lista de contatos e SMS's.

Já na seção 4.3, a memória volátil não foi examinada no segundo cenário, pois como também se trata de um emulador, o procedimento de compilação e captura da memória é exatamente o mesmo realizado na seção 4.2. Analisou-se nessa seção a memória persistente do dispositivo, com as ferramentas TSK e *Autopsy* e também foram coletadas informações extras importantes com o auxílio do *carving* de arquivos utilizando as ferramentas *Scalpel* e *Foremost*. Esperou-se que, com a análise de duas partições essenciais do Android, */data* e o cartão SD, fossem extraídos dados relevantes, com relação ao usuário e aos aplicativos utilizados pelo mesmo.

Com o intuito de validar as técnicas e ferramentas vistas nos capítulos anteriores, o último cenário, da seção 4.4, foi de um estudo de caso real. Apenas a memória persistente foi analisada, pois

esperou-se encontrar informações armazenadas pelo usuário no dispositivo, que não são perdidas mesmo quando o aparelho é desligado (o que não acontece com a memória volátil). Estas informações vão desde mensagens de texto, ligações, até detalhes armazenados em aplicativos muito utilizados, como *WhatsApp*, *Facebook*, *Snapchat*, entre outros.

4.2 Primeiro cenário - Emulador Nexus 5, Versão do Android 4.0.3

Como se viu no Capítulo 3, para utilizar o LiME é preciso compilar seu módulo no dispositivo alvo. Um dos problemas ao lidar com um *Android Virtual Device* - AVD, é que o *kernel* do mesmo não permite o uso de LKM's, assim, um novo *kernel* tem de ser compilado no dispositivo emulado para que o aparelho aceite o uso do módulo implementado no LiME. Para compilar o *kernel*, é preciso baixar o código-fonte correto, correspondente ao dispositivo em análise. Como analisado, neste cenário com o AVD, o código-fonte a ser verificado e que é usado por qualquer emulador Android, independentemente da versão, é o chamado *Goldfish*. A seguir, serão apresentados os passos necessários para a compilação do novo *kernel* [Chung].

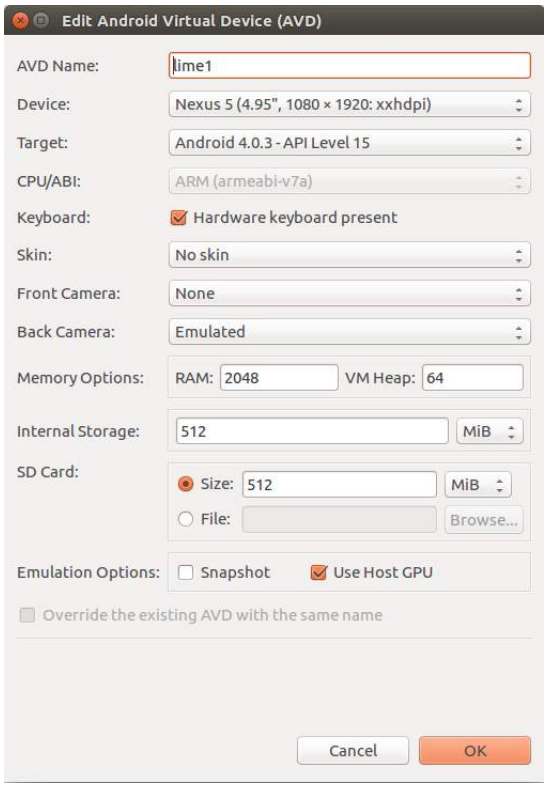


Figura 4.1: Configuração do primeiro emulador a ser analisado

O primeiro passo será a criação do emulador. Na Figura 4.1 são apresentadas as configurações que foram determinadas no momento da criação do AVD. As opções de quantidade de memória interna e do cartão de memória, ficam a critério do analista. As orientações específicas de como criar um AVD, a partir do Android Studio instalado, encontram-se no Anexo II.

O próximo passo é determinar a versão do *kernel* do emulador criado. Para isso é preciso inicializá-lo a fim de que seja acessado o *shell* (terminal onde é possível obter informações diversas do dispositivo). A inicialização é realizada da seguinte maneira:

```
emulator -avd lime1
```

Utilizando-se das instruções dos principais comandos do ADB contidas no Anexo II, é feito o procedimento da Figura 4.2.

```
raquel@raquel:~$ adb shell
adb server is out of date.  killing...
* daemon started successfully *
# cat /proc/version
Linux version 2.6.29-g46b05b2 (vchtchetkine@vc-irv.irv.corp.google.com) (gcc ver
sion 4.4.3 (GCC) ) #28 Thu Nov 17 06:39:36 PST 2011
#
```

Figura 4.2: Obtenção da versão do *kernel* do dispositivo

A versão utilizada é representada pelos dígitos após a letra 'g' da versão do *kernel*, neste caso, é a **46b05b2**, isto será importante para determinar a versão exata a ser compilada, para que não haja imprevistos. Para baixar o módulo *goldfish*, podemos utilizar o recurso *git* do Linux, como usado na Figura 4.3.

```
raquel@raquel:/usr/src/goldfish$ sudo git clone https://android.googlesource.com
/kernel/goldfish.git goldfish4
Cloning into 'goldfish4'...
remote: Sending approximately 1011.44 MiB ...
remote: Counting objects: 77879, done
remote: Finding sources: 100% (735/735)
remote: Total 4533936 (delta 3793122), reused 4533922 (delta 3793122)
Receiving objects: 100% (4533936/4533936), 1011.36 MiB | 1.96 MiB/s, done.
Resolving deltas: 100% (3793122/3793122), done.
Checking connectivity... done.
raquel@raquel:/usr/src/goldfish$ cd goldfish4/
raquel@raquel:/usr/src/goldfish/goldfish4$ sudo git checkout 46b05b2
[sudo] password for raquel:
Checking out files: 100% (26801/26801), done.
Note: checking out '46b05b2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 46b05b2... goldfish: Enable CONFIG_TUN
raquel@raquel:/usr/src/goldfish/goldfish4$
```

Figura 4.3: *Download* do *kernel Goldfish* para o emulador

Na Figura 4.3, o parâmetro *goldfish4* é o local em que o módulo será salvo e faz alusão à primeira versão do Android, 4.0.3, a ser analisada. É perceptível que o *kernel* foi alterado de uma versão geral do *Goldfish* para a versão específica, neste caso a **46b05b2**, encontrada anteriormente. Depois que o *download* foi feito, o *kernel* precisa ser compilado mas, antes, um arquivo específico

que será a principal parte da compilação precisa ser criado e depois editado. A Figura 4.4 mostra como este arquivo é criado.

```
raquel@raquel:/usr/src/goldfish/goldfish4$ sudo make ARCH=arm goldfish_armv7_def
config
arch/arm/configs/goldfish_armv7_defconfig:292:warning: override: FB_EARLYSUSPEND
 changes choice state
#
# configuration written to .config
#
raquel@raquel:/usr/src/goldfish/goldfish4$ sudo gedit .config
```

Figura 4.4: Criaçao do arquivo *.config*

Quando o arquivo estiver aberto para edição, a opção *CONFIG_ MODULES* deve ser procurada, pois, ela estará configurada com o valor padrão *"not set"*, e precisa ser modificada para que permita a configuração de módulos. Esta modificação pode ser feita com o acréscimo do parâmetro 'y', *CONFIG_ MODULES=y*. As Figuras 4.5 e 4.6 demonstram essa troca de parâmetros.

```
CONFIG_HAVE_OPROFILE=y
CONFIG_HAVE_KPROBES=y
CONFIG_HAVE_KRETPROBES=y
CONFIG_HAVE_GENERIC_DMA_COHERENT=y
CONFIG_SLABINFO=y
CONFIG_RT_MUTEXES=y
CONFIG_BASE_SMALL=0
# CONFIG_MODULES is not set
CONFIG_BLOCK=y
# CONFIG_LBD is not set
# CONFIG_BLK_DEV_IO_TRACE is not set
# CONFIG_BLK_DEV_BSG is not set
# CONFIG_BLK_DEV_INTEGRITY is not set
```

Figura 4.5: Parte do arquivo em que a opção *CONFIG_ MODULE* deve ser modificada

```
CONFIG_HAVE_OPROFILE=y
CONFIG_HAVE_KPROBES=y
CONFIG_HAVE_KRETPROBES=y
CONFIG_HAVE_GENERIC_DMA_COHERENT=y
CONFIG_SLABINFO=y
CONFIG_RT_MUTEXES=y
CONFIG_BASE_SMALL=0
CONFIG_MODULES=y
CONFIG_BLOCK=y
# CONFIG_LBD is not set
# CONFIG_BLK_DEV_IO_TRACE is not set
# CONFIG_BLK_DEV_BSG is not set
# CONFIG_BLK_DEV_INTEGRITY is not set
```

Figura 4.6: Modificação da opção *CONFIG_ MODULE*

Antes de efetuar a compilação propriamente dita, um passo extra deve ser executado. Em alguns testes, um erro era recorrente na compilação, por isso, uma alteração foi feita. O caminho de acesso até o diretório é: */usr/src/goldfish/goldfish4/kernel* (ou onde o analista tiver instalado o *kernel* do dispositivo), modificando o arquivo *timesconst.pl*.

Com o arquivo aberto, a modificação deve ser feita na linha 373. Esta apresentará a condição *if(!defined(@val))* e deve ser alterada para *if(!@val)* [Freetz]. A modificação está descrita nas

Figuras 4.7 e 4.8.

```
372         @val = @{$canned_values{$hz}};
373         if (!defined(@val)) {
374                 @val = compute_values($hz);
375         }
376         output($hz, @val);
```

Figura 4.7: Parte do arquivo *timeconst.pl* a ser modificada

```
372         @val = @{$canned_values{$hz}};
373         if (!@val) {
374                 @val = compute_values($hz);
375         }
376         output($hz, @val);
```

Figura 4.8: Modificação finalizada no arquivo *timeconst.pl*

Após isso, é preciso sair do diretório *kernel* (*cd - -*) e, finalmente, executar o procedimento para a compilação do *kernel*. Este está descrito na Figura 4.9, em que é necessário observar que o diretório informado em *CROSS_ COMPILE* deve ser o lugar onde está instalado o NDK (procedimento de instalação descrito no Anexo I).

```
raquel@raquel:/usr/src/goldfish/goldfish4$ sudo make ARCH=arm CROSS_COMPILE=/hom
e/raquel/Android/Sdk/ndk-bundle/toolchains/arm-linux-androideabi-4.9/prebuilt/li
nux-x86_64/bin/arm-linux-androideabi-
  CHK     include/linux/version.h
make[1]: 'include/asm-arm/mach-types.h' is up to date.
  CHK     include/linux/utsrelease.h
  UPD     include/linux/utsrelease.h
  SYMLINK include/asm -> include/asm-arm
  CALL    scripts/checksyscalls.sh
```

Figura 4.9: Execução do comando para compilar o *kernel*

O processo anterior estará correto se, ao final da compilação, o resultado seja o que está identificado na Figura 4.10, onde podem ser observados dois arquivos que são criados e que serão utilizados posteriormente, o *System.map* e o *zImage*.

```
  SYSMAP  System.map
  SYSMAP  .tmp_System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  GZIP    arch/arm/boot/compressed/piggy.gz
  AS      arch/arm/boot/compressed/piggy.o
  CC      arch/arm/boot/compressed/misc.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  Building modules, stage 2.
  MODPOST 1 modules
  CC      drivers/hid/hid-dummy.mod.o
  LD [M]  drivers/hid/hid-dummy.ko
raquel@raquel:/usr/src/goldfish/goldfish4$
```

Figura 4.10: Final da compilação com resultados esperados

Para utilizar o *kernel* compilado no dispositivo emulado, deve-se inicializá-lo de uma maneira

diferente, que está descrita na Figura 4.11. É interessante observar o uso do arquivo *zImage* que foi criado no processo de compilação.

```
raquel@raquel:~$ emulator -avd lime1 -kernel /usr/src/goldfish/goldfish4/arch/ar
m/boot/zImage
emulator: WARNING: Classic qemu does not support SMP. The hw.cpu.ncore option fr
om your config file is ignored.
bind: Transport endpoint is not connected
emulator: Listening for console connections on port: 5556
emulator: Serial number of this emulator (for ADB): emulator-5556
```

Figura 4.11: Comando para inicialização do emulador com o *kernel* compilado

Depois da finalização do procedimento com o *kernel*, pode-se compilar o LiME e o *Volatility* para analisar a memória volátil do dispositivo. Para isso, após instalar as duas ferramentas (Anexo I), deve-se modificar o arquivo *Makefile* (que é criado em formato padrão no momento da instalação) de ambos para que se consiga encontrar os diretórios corretos do *kernel* e do elemento de *Cross-Compile*. O *Makefile* do LiME está localizado dentro da pasta *src*, onde foi instalada a ferramenta, enquanto o do *Volatility* está localizado em */volatility/tools/linux*. Os arquivos modificados que foram utilizados encontram-se no Anexo IV.

Outra modificação necessária é no arquivo *main.c*, que é obtido diretamente do *download* da ferramenta LiME. O arquivo obtido a partir da instalação, utiliza uma certa função que, nos testes feitos neste projeto, produziu um erro no momento da compilação. Por isso, utiliza uma versão modificada do mesmo e esta se encontra no Anexo IV. Após a alteração de todos os arquivos, deve-se compilar cada uma das ferramentas, de acordo com as Figuras 4.12, 4.13 e 4.14.

```
raquel@raquel:~$ cd /usr/src/goldfish/LiME4/src/
raquel@raquel:/usr/src/goldfish/LiME4/src$ sudo make
[sudo] password for raquel:
make ARCH=arm CROSS_COMPILE=/home/raquel/Android/Sdk/ndk-bundle/toolchains/arm-l
inux-androideabi-4.9/prebuilt/linux-x86_64/bin/arm-linux-androideabi- EXTRA_CFLA
GS=-fno-pic -C /usr/src/goldfish/goldfish4/ M=/usr/src/goldfish/LiME4/src module
s
make[1]: Entering directory '/usr/src/goldfish/goldfish4'
  CC [M]  /usr/src/goldfish/LiME4/src/tcp.o
  CC [M]  /usr/src/goldfish/LiME4/src/disk.o
  CC [M]  /usr/src/goldfish/LiME4/src/main.o
/usr/src/goldfish/LiME4/src/main.c: In function 'write_range':
/usr/src/goldfish/LiME4/src/main.c:205:5: warning: 'return' with a value, in fun
ction returning void
     return (int) s;
     ^
/usr/src/goldfish/LiME4/src/main.c:213:2: warning: 'return' with a value, in fun
ction returning void
  return 0;
  ^
  LD [M]  /usr/src/goldfish/LiME4/src/lime.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /usr/src/goldfish/LiME4/src/lime.mod.o
  LD [M]  /usr/src/goldfish/LiME4/src/lime.ko
make[1]: Leaving directory '/usr/src/goldfish/goldfish4'
mv lime.ko lime-goldfish_4.ko
raquel@raquel:/usr/src/goldfish/LiME4/src$
```

Figura 4.12: Compilação da ferramenta LiME

Para analisar a memória coletada com a ferramenta *Volatility* é preciso criar um perfil para

```
raquel@raquel:/usr/src/goldfish$ cd volatility4/tools/linux/
raquel@raquel:/usr/src/goldfish/volatility4/tools/linux$ sudo make
make ARCH=arm CROSS_COMPILE=/home/raquel/Android/Sdk/ndk-bundle/toolchains/arm-l
inux-androideabi-4.9/prebuilt/linux-x86_64/bin/arm-linux-androideabi- -C /usr/sr
c/goldfish/goldfish4 \ CONFIG_DEBUG_INFO=y M= modules
make[1]: Entering directory '/usr/src/goldfish/goldfish4'
  CHK     include/linux/version.h
make[2]: 'include/asm-arm/mach-types.h' is up to date.
  CHK     include/linux/utsrelease.h
  SYMLINK include/asm -> include/asm-arm
  CC      kernel/bounds.s
  GEN     include/linux/bounds.h
  CC      arch/arm/kernel/asm-offsets.s
  GEN     include/asm/asm-offsets.h
  CALL    scripts/checksyscalls.sh
```

Figura 4.13: Compilação da ferramenta *Volatility*

```
  HOSTCC  scripts/mod/sumversion.o
  HOSTLD  scripts/mod/modpost
  CC [M]  drivers/hid/hid-dummy.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      drivers/hid/hid-dummy.mod.o
  LD [M]  drivers/hid/hid-dummy.ko
make[1]: Leaving directory '/usr/src/goldfish/goldfish4'
dwarfdump -di module.ko > module_4.dwarf
raquel@raquel:/usr/src/goldfish/volatility4/tools/linux$
```

Figura 4.14: Final da Compilação da ferramenta *Volatility*

cada dispositivo. Na Figura 4.15 apresenta-se como este perfil é gerado. É importante observar que neste momento, são utilizados o módulo *dwarf* criado pela compilação do *Volatility* e também o arquivo *System.map*, que foi criado na compilação do *kernel*.

```
raquel@raquel:/usr/src/goldfish/volatility4/tools/linux$ zip -j Android_Goldfish
4.zip module_4.dwarf /usr/src/goldfish/goldfish4/System.map
  adding: module_4.dwarf (deflated 89%)
  adding: System.map (deflated 73%)
raquel@raquel:/usr/src/goldfish/volatility4/tools/linux$ cp Android_Goldfish4.zi
p /usr/src/goldfish/
raquel@raquel:/usr/src/goldfish/volatility4/tools/linux$ cp Android_Goldfish4.zi
p /usr/src/goldfish/volatility4/volatility/plugins/overlays/linux/
raquel@raquel:/usr/src/goldfish/volatility4/tools/linux$
```

Figura 4.15: Criação do perfil do dispositivo a ser analisado

O próximo passo foi transferir o arquivo *lime-goldfish4.ko*, gerado na compilação do LiME, para o emulador analisado. Primeiramente, assim como foi feito no momento de se descobrir a versão do dispositivo, o emulador precisa estar inicializado para que o acesso por meio do ADB esteja disponível. **É necessário lembrar que o emulador tem de ser executado com o *kernel* que foi compilado!** Com o AVD iniciado o procedimento de envio pode ocorrer e, logo depois, a inicialização do módulo, através do comando *insmod*. Estes processos são demonstrados na Figura 4.16. Durante este processo, é preciso ter em mente as opções de armazenamento da memória e de transferência da informação, coletadas e discutidas no Capítulo 3, na Seção 3.2.

Após este último comando, *insmod*, o dispositivo entrará em modo de espera para que consiga transferir os dados. Como neste caso foi utilizada a transferência por meio de conexão TCP,

```
raquel@raquel:~$ adb push /usr/src/goldfish/LiME4/src/lime-goldfish_4.ko /sdcard
/lime.ko
153 KB/s (7876 bytes in 0.049s)
raquel@raquel:~$ adb forward tcp:4444 tcp:4444
raquel@raquel:~$ adb shell
# insmod /sdcard/lime.ko "path=tcp:4444 format=lime"
```

Figura 4.16: Procedimento de envio e inicialização do módulo do LiME para o dispositivo

para que fosse concluída é preciso que, em outro terminal, no computador do investigador, o procedimento (uso do *netcat*) da Figura 4.17 seja realizado.

```
raquel@raquel:~$ nc localhost 4444 > goldfish4.lime
```

Figura 4.17: Uso do *netcat* para a transferência do *dump* da memória

Na Figura 4.17, depois do sinal de maior (>) utilizado no comando, insere-se o nome desejado para o arquivo que será salvo e sua respectiva extensão, que neste caso foi *.lime*. Após alguns minutos, dependendo do tamanho da memória coletada, o arquivo será gerado e salvo diretamente no computador *host*. Com o perfil criado, pode-se, então, analisar a memória com o *Volatility*.

A partir desta memória, vários testes e descobertas importantes com relação ao dispositivo foram realizados. Depois de se conhecer todos os *plugins* do *Volatility*, definidos no Capítulo 3, Seção 3.3, que retornam informações úteis para o sistema operacional Android, pode-se aplicá-los para que sua eficiência seja comprovada em relação a outras soluções. Antes de começar as análises, é importante definir algumas variáveis de ambiente, que facilitam o uso do *Volatility Framework*. Estas definições são descritas na Figura 4.18.

```
raquel@raquel:~$ cd /usr/src/goldfish/volatility4/
raquel@raquel:/usr/src/goldfish/volatility4$ export VOLATILITY_LOCATION=file:///
home/raquel/goldfish4.lime
raquel@raquel:/usr/src/goldfish/volatility4$ export VOLATILITY_PROFILE=LinuxAndr
oid_Goldfish4ARM
raquel@raquel:/usr/src/goldfish/volatility4$
```

Figura 4.18: Uso de variáveis de ambiente que facilitam as análises com o *Volatility*

4.2.1 Análise de permissões em aplicativos instalados no dispositivo

Depois de determinar as variáveis da Figura 4.18, a primeira análise a ser feita é com relação às permissões dos aplicativos presentes no momento do *dump* da memória do dispositivo. Cada aplicação requer permissões no dispositivo, estas são adicionadas ao arquivo *AndroidManifest.xml* presente em qualquer aplicativo instalado. O gerenciamento destas permissões pode ser feito analisando-se o arquivo *packages.xml*, presente em todos os dispositivos Android [Sylve].

Estas permissões podem ser analisadas com o *plugin linux_find_file* do *Volatility Framework*. Este, usualmente, é utilizado em duas etapas. Na primeira, é necessário encontrar o *inode* do

arquivo. O *inode* é uma estrutura responsável por obter as informações básicas do arquivo, como permissões de acesso ao mesmo, dados referentes ao último acesso, alterações, tamanho do arquivo e também os ponteiros para o arquivo em si. Em resumo, pode-se considerar o *inode* como a identidade do arquivo, sendo esta uma identificação única. Para encontrar esta identificação, utiliza-se o *plugin* como está descrito na Figura 4.19.

```
raquel@raquel:/usr/src/goldfish/volatility4$ python vol.py linux_find_file -F /d
ata/system/packages.xml
Volatility Foundation Volatility Framework 2.5
Inode Number          Inode File Path
--------------- ---------- ---------
           1064 0xf35a1ab8 /data/system/packages.xml
raquel@raquel:/usr/src/goldfish/volatility4$ python vol.py linux_find_file -i 0x
f35a1ab8 -O packages.xml
Volatility Foundation Volatility Framework 2.5
```

Figura 4.19: Uso do *plugin linux_find_file*

A partir da Figura 4.19 infere-se que o *inode* do arquivo *packages.xml* é **0xf34c4d08**. Esta identificação será importante para o próximo passo, em que será possível exportar o arquivo com as informações sobre os aplicativos. Para isso, é preciso que se faça uso do parâmetro *-i/–inode* para que os arquivos em *cache* na memória sejam exportados. O parâmetro *-O* também é utilizado e especifica onde os resultados do arquivo serão salvos. No caso da Figura 4.19, o final do comando utilizado, *packages.xml*, é o nome do arquivo em que serão salvas as informações coletadas do dispositivo.

O arquivo *packages.xml*, como explicado, tem o papel de armazenar as permissões dadas por cada aplicativo do dispositivo. Ao analisar o arquivo salvo, percebe-se que, primeiramente, são informados alguns dados gerais do dispositivo, como o número da plataforma (que determina a versão do Android instalada no mesmo) e também seu código de verificação. Após estas informações gerais, apresentam-se as permissões que estão presentes no dispositivo, tanto aquelas que são nativas do próprio Android, como as específicas de certos aplicativos instalados. A Figura 4.20 mostra um exemplo de como estas permissões gerais são informadas.

```
<packages>
  <last-platform-version internal="15" external="15"/>
  <verifier device="E5QG-JYLT-F33D-X"/>
  <permission-trees/>
- <permissions>
    <item name="android.permission.CHANGE_WIFI_MULTICAST_STATE" package="android" protection="1"/>
    <item name="org.mozilla.firefox_sync.permission.PER_ACCOUNT_TYPE" package="org.mozilla.firefox" protection="2"/>
    <item name="com.facebook.receiver.permission.ACCESS" package="com.facebook.orca" protection="2"/>
    <item name="android.permission.CLEAR_APP_CACHE" package="android" protection="1"/>
```

Figura 4.20: Identificação das permissões do dispositivo

Na Figura 4.20 identificam-se duas permissões do sistema Android, a *CLEAR_APP_CACHE* e *CHANGE_WIFI_MULTICAST_STATE* e, além destas, também identificam-se duas permissões especiais dos aplicativos *Mozilla Firefox* e *Facebook*. No final da linha de cada item de permissão, existe a opção *protection*, que é identificada por um número de 1 a 4, que apresentará o nível de proteção de uma permissão específica. Estes níveis foram apresentados no Capítulo 2.

A parte mais importante ao analisar o arquivo *packages.xml* foi o exame de aplicativos em que

pudessem ser considerados como tipos de *malware*. Isto pode ser feito vendo as permissões que cada um destes requisitou ao dispositivo. Para exemplificar esta situação de risco, foram utilizadas duas amostras de aplicativos que atuaram como *malware* [Parkour].

Foram escolhidos para análise os *apps Angry Birds*, que é um jogo, e o do Banco do Brasil, ambos modificados para agirem de forma maliciosa no dispositivo. Na Figura 4.21 pode-se observar que o ícone dos dois aplicativos passam despercebidos como se fossem originais. No caso do jogo, o original apresenta um símbolo no canto superior direito do ícone da empresa que o desenvolve e distribui, a *Rovio*. E no caso do Banco do Brasil, além do original ter uma leve diferença de cor, o nome original é a abreviatura do nome do banco, BB. Como é possível perceber na imagem a seguir, os *apps* maliciosos estão destacados em vermelho.

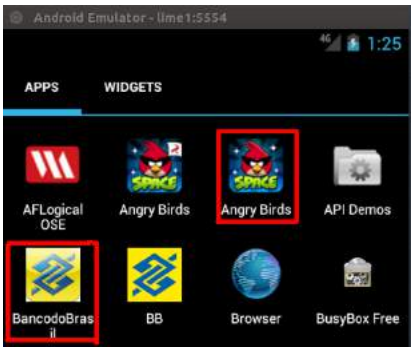


Figura 4.21: Comparação entre os ícones dos aplicativos

No arquivo *packages.xml* coletado com o *plugin*, pode-se identificar os tipos de permissões que foram concedidas aos aplicativos para, então, serem confirmados quais deles terão chances de apresentarem caráter malicioso. A Figura 4.22 mostra as permissões do aplicativo falso do Banco do Brasil. O primeiro item estranho nas informações do *app* é o nome que foi dado, *appinventor.ai_funayamajogos.Bancodo-Brasil*. Além disso, como visto no Capítulo 2, uma das permissões concedidas ao *app* foi a de ter acesso à função de modificar ou deletar conteúdos do cartão de memória (*WRITE_EXTERNAL_ STORAGE*), que faz parte do grupo de permissões perigosas.

```
−<package name="appinventor.ai_funayamajogos.BancodoBrasil" codePat
 /data/appinventor.ai_funayamajogos.BancodoBrasil/lib" flags="0" ft="158a
  −<sigs count="1">
     <cert index="5"
     key="3082034130820229a003020102020426506 87b300d06092a86488
   </sigs>
  −<perms>
     <item name="android.permission.READ_PHONE_STATE"/>
     <item name="android.permission.ACCESS_WIFI_STATE"/>
     <item name="android.permission.ACCESS_NETWORK_STATE"/>
     <item name="android.permission.INTERNET"/>
     <item name="android.permission.WRITE_EXTERNAL_STORAGE"/>
   </perms>
 </package>
```

Figura 4.22: Identificação das permissões do aplicativo *Banco do Brasil*

A Figura 4.23 mostra as permissões concedidas ao *app* falso do jogo *Angry Birds*. Diferentemente do *app* do banco, este não apresenta um nome suspeito, pelo contrário, mostra, inclusive, a empresa responsável por fazer o jogo, a *Rovio*. Com relação às permissões, apresenta duas de caráter perigoso, a mesma do exemplo anterior e também a permissão que faz com que o *app* leia vários arquivos de *log* do sistema (*READ_LOGS*), o que possibilita que se descubram informações gerais sobre o que o usuário faz com o dispositivo, o que, potencialmente, pode incluir informações pessoais ou privadas.

Figura 4.23: Identificação das permissões do aplicativo *Angry Birds*

Outro fator que pode ser observado em ambos os casos, é que nenhum dos dois *apps* possuía um tipo de permissão específica destinada somente para eles, como usualmente acontece nos aplicativos. A partir das informações encontradas acima, já é possível passar a investigar de maneira mais específica as atividades do *app* que está instalado no dispositivo. Para tanto, o investigador pode procurar pelo arquivo *.apk* do aplicativo, com o auxílio do mesmo *plugin*, *linux_find_file*, e então testar o aplicativo em alguma ferramenta de análise de *malware*. A Figura 4.24 mostra como obter o arquivo *.apk* do jogo analisado. Pode-se utilizar, por exemplo, o *MobSF* ou, de maneira mais rápida, o *site VirusTotal* que é um serviço gratuito que analisa arquivos e URLs suspeitas e facilita a rápida detecção de vírus, *worms*, *trojans*, e todos os tipos de *malwares* [VirusTotal].

Figura 4.24: Captura do arquivo *.apk* do aplicativo de jogo analisado

4.2.2 Análise de diretórios e arquivos temporários

Outra análise a ser feita é a recuperação de arquivos *tmpfs*. O propósito do *plugin linux_tmpfs* é o de reconstruir qualquer arquivo de sistema *tmpfs* contido em uma captura de memória. O

tmpfs é um tipo de arquivo de sistema que está contido apenas na memória RAM do dispositivo. Isto significa que arquivos e diretórios presentes em partições do tipo *tmpfs* nunca são escritas no disco local e uma vez que a partição é desmontada, o arquivo de sistema inteiro é perdido [Case].

Em uma perspectiva forense, o *tmpfs* é interessante por alguns motivos. O primeiro é que, em um cenário tradicional, quando um dispositivo é desligado, o esperado é que quando se faça a imagem do mesmo, seja possível recuperar os arquivos de sistema no momento em que o aparelho estava ligado. Porém, como visto, isto não é possível. Quando falamos do *tmpfs*, tudo seria perdido. O segundo motivo é que ele pode ser utilizado em sistemas baseados em Linux, por atacantes e certos tipos de *malwares*. Em vários sistemas o diretório */tmp* é montado como *tmpfs*, porque, dessa maneira, o diretório será liberado após um *reboot* do dispositivo. Este diretório, geralmente, contém dados de rascunho e instala arquivos para várias aplicações assim como para *rootkits* e *malwares* que tentam ser inperceptíveis para o sistema em que é instalado [Case].

Distribuições que utilizam Linux também usam o *tmpfs* para implementar memórias compartilhadas através do diretório */dev/shm*. Este diretório também pode ser utilizado por atacantes para fazer o *download* de arquivos, compilar programas e armazenar saídas de comando e *malware hooks*. Como arquivos de sistema são perdidos após *reboots*, os atacantes sabem que podem frustrar e derrotar investigações forenses tradicionais usando este método. As várias aplicações no Android usam instâncias *tmpfs*, criadas em tempo de execução, para armazenar dados interessantes de tempo, *status* do tempo de execução – o *runtime* [Case].

Assim como o *plugin* apresentado anteriormente, o *linux_ tmpfs* é utilizado em duas etapas. A primeira, é a listagem de todos os sistemas de arquivos *tmpfs* que estão presentes na imagem da memória que foi coletada. Isto é feito acrescentando-se o parâmetro *-L* após o uso do comando. A segunda etapa é a de recuperar os arquivos. E para recuperar qualquer um dos sistemas de arquivos encontrados, é preciso usar o parâmetro *-S*, o número atribuído ao sistema alvo e também dar um diretório de saída para que seja salvo (com o parâmetro *-D*). Por exemplo, se quisermos salvar o conteúdo do sistema de arquivos */mnt/sdcard/.android_ secure* devemos seguir o procedimento da Figura 4.25, que mostra as duas etapas deste processo.

```
raquel@raquel:/usr/src/goldfish/volatility4$ python vol.py linux_tmpfs -L
Volatility Foundation Volatility Framework 2.5
1 -> /dev
2 -> /mnt/obb
3 -> /mnt/sdcard/.android_secure
4 -> /mnt/asec
raquel@raquel:/usr/src/goldfish/volatility4$ python vol.py linux_tmpfs -S 2 -D A
ndroid/
Volatility Foundation Volatility Framework 2.5
```

Figura 4.25: Uso do *plugin linux_ tmpfs*

Quando o comando é finalizado, todo o sistema de arquivos, incluindo sub-diretórios, e *metadados*, serão replicados para o diretório *Android/* indicado (o diretório foi criado na máquina hospedeira). Em uma investigação forense real, o local em que serão salvas as informações é, preferencialmente, um dispositivo externo.

4.2.3 Análise de divulgação de informações de usuários por aplicativos

A próxima análise serve para examinar o quanto um aplicativo é seguro, com relação a não divulgar dados importantes relacionados ao usuário (seu nome, senhas, entre outros). Para encontrar este tipo de informação, para um aplicativo específico, primeiro é preciso localizar a identificação única do processo (PID), então este é mapeado na memória para descobrir os *offsets* do *heap*, que é a área da memória que é de interesse e, por último, é feito o *dump* do *heap* [Stirparo, Fovino e Kounelis 2013]. Para isso, utilizam-se três *plugins* do *Volatility*:

- *linux_pslist*;
- *linux_proc_maps*;
- *linux_dump_map*.

Por exemplo, foi analisado o aplicativo *Facebook*. Para isso, foi utilizado o primeiro *plugin* para listar todos os processos. As Figuras 4.26 e 4.27 mostram este procedimento. A lista foi reduzida por conter muitos processos e informações. A partir da Figura 4.27, identifica-se o PID do *app*, que neste caso teve o valor de '311'.

```
raquel@raquel:/usr/src/goldfish/volatility4$ python vol.py linux_pslist
Volatility Foundation Volatility Framework 2.5
Offset     Name                 Pid             PPid            Uid
Gid    DTB        Start Time
---------- -------------------- --------------- --------------- ---------------
------ ---------- ----------
0xf3812c00 init                 1               0               0
0      0x33b04000 2016-11-20 22:38:11 UTC+0000
0xf3812800 kthreadd             2               0               0
0      ---------- 2016-11-20 22:38:11 UTC+0000
0xf3812400 ksoftirqd/0          3               2               0
0      ---------- 2016-11-20 22:38:11 UTC+0000
0xf3812000 events/0             4               2               0
0      ---------- 2016-11-20 22:38:11 UTC+0000
0xf3819c00 khelper              5               2               0
0      ---------- 2016-11-20 22:38:11 UTC+0000
0xf3819800 suspend              6               2               0
0      ---------- 2016-11-20 22:38:11 UTC+0000
```

Figura 4.26: Identificação dos processos com o *plugin linux_pslist*

Após a identificação do PID, usa-se o segundo *plugin* (*linux_proc_maps*) para encontrar os *offsets* específicos do *app*. A Figura 4.28 mostra como obter os *offsets* do processo analisado.

É preciso fazer o *dump* tanto do *heap* principal do *app*, quanto o *heap* da *DalvikVM*. A razão da existência dessa quantidade de *heaps* é porque, quando uma aplicação é iniciada, ela é executada em seu próprio processo, que contém a sua instância específica. Portanto, um *heap* é nativo do processo em si e o outro é o *heap* interno da máquina virtual. Como a aplicação é executada dentro da *DalvikVM*, é esperado que sejam encontrados dados de entrada inseridos pelo usuário no *heap* da *DalvikVM*, sendo possível até que sejam encontrados nos dois.

O *dump* do primeiro *offset* é identificado na Figura 4.29. Neste primeiro comando foi feito o *dump* do *heap* nativo (endereço 0x000000000000b000). O *plugin* extrai informações de todos os processos, portanto, a imagem foi reduzida.

```
0xeaf39c00 facebook.katana      311             39              -
10043  0x2af2c000 2016-11-20 22:39:45 UTC+0000
0xeafa3c00 ainfire.supersu      361             39              -
10047  0x2185c000 2016-11-20 22:39:54 UTC+0000
0xe18ac800 ndroid.contacts      376             39              -
10006  0x218c8000 2016-11-20 22:39:54 UTC+0000
0xe18e9c00 com.android.mms      395             39              10000
10000  0x21948000 2016-11-20 22:39:54 UTC+0000
0xe18ee400 d.process.media      407             39              -
10007  0x2199c000 2016-11-20 22:39:54 UTC+0000
0xe19ec800 ndroid.exchange      437             39              -
10001  0x21a24000 2016-11-20 22:39:55 UTC+0000
0xe1a66400 m.android.email      454             39              -
10004  0x21a80000 2016-11-20 22:39:55 UTC+0000
0xe1adf400 mozilla.firefox      480             39              -
10040  0x21b08000 2016-11-20 22:39:56 UTC+0000
0xeaf97000 sh                   642             47              0
0      0x1e074000 2016-11-20 22:53:27 UTC+0000
0xeaf97400 insmod               644             642             0
0      0x21bf0000 2016-11-20 22:53:43 UTC+0000
```

Figura 4.27: Identificação dos processos com o *plugin linux_ pslist*

```
raquel@raquel:/usr/src/goldfish/volatility4$ python vol.py linux_proc_maps -p 31
1 | grep heap
Volatility Foundation Volatility Framework 2.5
0x00000000eaf39c00      311 facebook.katana      0x000000000000b000 0x0000000000
71f000 rw-           0x0      0      0          0 [heap]
0x00000000eaf39c00      311 facebook.katana      0x00000000409b1000 0x0000000042
693000 rw-           0x0      0      7        372 /dev/ashmem/dalvik-heap
0x00000000eaf39c00      311 facebook.katana      0x0000000042693000 0x0000000050
9b1000 ---     0x1ce2000      0      7        372 /dev/ashmem/dalvik-heap
0x00000000eaf39c00      311 facebook.katana      0x0000000058e54000 0x0000000058
e55000 r--           0x0      0      7        380 /dev/ashmem/SurfaceFlinger rea
d-only heap
```

Figura 4.28: Uso do *plugin linux_ proc_ maps* para encontrar os *offsets*

```
raquel@raquel:/usr/src/goldfish/volatility4$ python vol.py linux_dump_map -s 0x0
00000000000b000 --dump-dir /home/raquel/face-heap/
Volatility Foundation Volatility Framework 2.5
Task       VM Start   VM End         Length Path
---------- ---------- ---------- ---------- ----
        32 0x0000b000 0x0000c000     0x1000 /home/raquel/face-heap/task.32.0xb00
0.vma
        37 0x0000b000 0x0000e000     0x3000 /home/raquel/face-heap/task.37.0xb00
0.vma
        39 0x0000b000 0x001f0000   0x1e5000 /home/raquel/face-heap/task.39.0xb00
0.vma
        84 0x0000b000 0x0031f000   0x314000 /home/raquel/face-heap/task.84.0xb00
0.vma
       139 0x0000b000 0x00252000   0x247000 /home/raquel/face-heap/task.139.0xb0
00.vma
       153 0x0000b000 0x0021e000   0x213000 /home/raquel/face-heap/task.153.0xb0
00.vma
       171 0x0000b000 0x00212000   0x207000 /home/raquel/face-heap/task.171.0xb0
00.vma
```

Figura 4.29: *Dump* do primeiro *offset* com o *plugin linux_ dump_ map*

A Figura 4.30 mostra como foi feito o *dump* o *dump* do *heap* da *DalvikVM* (endereço 0x00000000 409b1000).

Assim como na listagem de processos, alguns *tasks* foram retirados, pois, havia muitos processos recorrentes do *dump*. Os arquivos salvos com o *dump* da memória são binários e foram analisados com o auxílio de um editor hexadecimal. Neste caso, foi utilizado o *Bless Hex Editor* [Hex]. A instalação do mesmo está descrita no Anexo I.

Na análise, foram observadas palavras-chave como: *username*, *password* e também diretamente

```
raquel@raquel:/usr/src/goldfish/volatility4$ python vol.py linux_dump_map -s 0x0
0000000409b1000 --dump-dir /home/raquel/face-heap/
Volatility Foundation Volatility Framework 2.5
Task       VM Start   VM End         Length Path
---------- ---------- ---------- ---------- ----
        39 0x409b1000 0x42043000  0x1692000 /home/raquel/face-heap/task.39.0x409
b1000.vma
        84 0x409b1000 0x42533000  0x1b82000 /home/raquel/face-heap/task.84.0x409
b1000.vma
       139 0x409b1000 0x43653000  0x2ca2000 /home/raquel/face-heap/task.139.0x40
9b1000.vma
       153 0x409b1000 0x42083000  0x16d2000 /home/raquel/face-heap/task.153.0x40
9b1000.vma
       171 0x409b1000 0x42063000  0x16b2000 /home/raquel/face-heap/task.171.0x40
9b1000.vma
       185 0x409b1000 0x42103000  0x1752000 /home/raquel/face-heap/task.185.0x40
9b1000.vma
       196 0x409b1000 0x42993000  0x1fe2000 /home/raquel/face-heap/task.196.0x40
9b1000.vma
```

Figura 4.30: *Dump* do segundo *offset* com o *plugin linux_ dump_ map*

com o nome de usuário que foi utilizado para o teste. Ao procurar pelo nome de usuário de *login* do *app* no *dump task.890.0xb000*, foi encontrada facilmente a informação. A Figura 4.31 mostra este resultado.

```
tempt_count/rabesidona2012@hotmail.com.1E...
s.'/account_confirmation/email_oauth_last_at
tempt_time.1478481452077.........g...J.....
```

Figura 4.31: Análise do *dump* da memória do aplicativo *Facebook*

Como se pode observar, no momento da captura da memória, o aplicativo apresentava informações sobre a conta de usuário. Neste caso, foi o *e-mail* (rabesidona2012@hotmail.com), utilizado como *login*, que pode ser observado no arquivo binário. Após esta análise, pode-se verificar que mais informações críticas podem ser adquiridas de um aplicativo, dependendo da sua segurança, com relação aos dados do usuário a serem protegidos. Portanto, na perspectiva da Forense de dispositivos móveis, dados que muitas vezes não podem ser colhidos diretamente do aplicativo, podem ser obtidos pela captura da memória do dispositivo, além de poder fazer análises sobre a segurança dos *apps*.

4.2.4 AFLogical OSE

Para que alguns testes fossem feitos com esta ferramenta, foi preciso simular algumas ligações e mensagens de textos recebidas e enviadas pelo dispositivo. Estas simulações são encontradas no Anexo III. Na Figura 4.32 mostra-se como instalar o arquivo *.apk* no dispositivo.

Após a instalação, o analista tem que acessar o dispositivo e executar o aplicativo *AFLogical OSE*. A Figura 4.33 mostra como ele se comporta ao ser executado.

Marcam-se as opções das quais se deseja obter dados e, então, a opção *Capture* é selecionada e, depois, confirma-se com um *OK* para que seja tudo salvo. A Figura 4.34 mostra onde as informações foram salvas no dispositivo. Geralmente, elas são armazenadas no cartão de memória,

```
raquel@raquel:~$ cd /home/raquel/AFLogical-OSE_1.5.2/
raquel@raquel:~/AFLogical-OSE_1.5.2$ ls
AFLogical-OSE_1.5.2.apk  GPL  README.txt
raquel@raquel:~/AFLogical-OSE_1.5.2$ adb install AFLogical-OSE_1.5.2.apk
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
399 KB/s (28794 bytes in 0.070s)
        pkg: /data/local/tmp/AFLogical-OSE_1.5.2.apk
Failure [INSTALL_FAILED_ALREADY_EXISTS]
rm failed for -f, Read-only file system
raquel@raquel:~/AFLogical-OSE_1.5.2$
```

Figura 4.32: Instalação do arquivo *.apk* no dispositivo

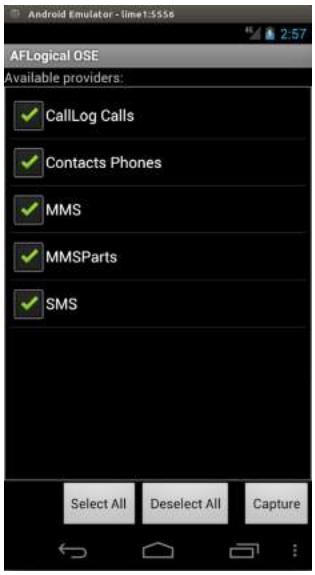


Figura 4.33: Comportamento da execução do aplicativo *AFLogical OSE* no dispositivo analisado

em uma pasta com o nome *forensics*.

```
raquel@raquel:~$ adb shell
# cd /sdcard
# ls
Android
DCIM
LOST.DIR
busybox-44.apk
forensics
git
lime.ko
media
mobile
# cd forensics
# ls
20161107.0345
20161127.2132
20161127.2133
#
```

Figura 4.34: Localização das informações coletadas

A Figura 4.35 mostra como a função *pull* do ADB foi utilizada para se transferir as informações coletadas do dispositivo para o computador do investigador.

```
raquel@raquel:~$ adb pull /sdcard/forensics/20161127.2133 /home/raquel
pull: building file list...
pull: /sdcard/forensics/20161127.2133/Contacts Phones.csv -> /home/raquel/Contac
ts Phones.csv
pull: /sdcard/forensics/20161127.2133/CallLog Calls.csv -> /home/raquel/CallLog
Calls.csv
pull: /sdcard/forensics/20161127.2133/MMS.csv -> /home/raquel/MMS.csv
pull: /sdcard/forensics/20161127.2133/MMSParts.csv -> /home/raquel/MMSParts.csv
pull: /sdcard/forensics/20161127.2133/SMS.csv -> /home/raquel/SMS.csv
pull: /sdcard/forensics/20161127.2133/info.xml -> /home/raquel/info.xml
6 files pulled. 0 files skipped.
97 KB/s (55739 bytes in 0.558s)
raquel@raquel:~$
```

Figura 4.35: Transferência das informações do dispositivo para o computador do investigador

Após a transferência de informações, é possível analisar os arquivos. Estes são salvos em formato *.csv* e os resultados são apresentados em tabelas. As Figuras 4.36, 4.37 e 4.38 mostram, respectivamente, o registro de chamadas do dispositivo, os contatos salvos e também o SMS que estava armazenado no mesmo.

A	B	C	D	E
_id	number	date	duration	type
1	123456	1478495936619	2	1
2	123456	1478495999197	24	1
3	123456	1478496043517	0	3

Figura 4.36: Resultado do registro de chamadas do dispositivo

Na Figura 4.36, são apresentadas as chamadas que foram recebidas ou efetuadas pelo dispositivo. A coluna *number* identifica o número do telefone celular que contactou ou foi contactado; a coluna *date* representa a data da chamada e está no formato *Epoch* do Linux (pode ser transformada utilizando [Epoch]); *duration* apresenta o tempo da ligação em segundos e *type* identifica se a chamada foi recebida (‘1’), efetuada (‘2’) ou perdida (‘3’).

A	B	C	D
person	number	display_name	name
1	1 234-56	Teste1	Teste1

Figura 4.37: Resultado do registro de contatos do dispositivo

Na Figura 4.37 é exibido o contato que estava salvo no dispositivo, mostrando o número na coluna *number* e o nome do mesmo em duas colunas, *display_ name* e *name*.

A	B	C	D	E
_id	address	date	type	body
1	123456	1478495888502	1	Este é um teste de SMS para um emulador Android.

Figura 4.38: Resultado do registro de SMS do dispositivo

Na Figura 4.38 é exibido o SMS armazenado, apresenta em *address* o número relacionado à mensagem; em *date* a data da mensagem, também no formato *Epoch*; a coluna *type* identifica se a mensagem foi recebida (‘1’) ou enviada (‘2’) e, por último, o conteúdo do SMS é apresentado na coluna *body*.

Estas informações coletadas pela ferramenta são de extrema importância em uma investigação, e a mesma pode ser utilizada num momento em que sejam necessários resultados rápidos e simples. Vale ressaltar que o dispositivo tem que estar em modo *root*, assim como quando estiver em uso por qualquer outra das ferramentas apresentadas nesta pesquisa.

4.3 Segundo cenário - Emulador Nexus 5, Versão do Android 6.0

O emulador foi inicializado com os comandos já conhecidos e que, como no primeiro cenário, podem ser vistos no Anexo II. Após isto, é necessário determinar quais partições do dispositivo serão extraídas para análise. Como observado no Capítulo 2, a partição */data* e também a do cartão de memória, são as de maior interesse para o investigador, já que os dados mais relevantes do dispositivo estão armazenados nestas partições. A Figura 4.39 mostra uma visão de todas as partições.

```
root@generic:/ # mount
rootfs / rootfs ro,seclabel,relatime 0 0
tmpfs /dev tmpfs rw,seclabel,nosuid,relatime,mode=755 0 0
devpts /dev/pts devpts rw,seclabel,relatime,mode=600 0 0
proc /proc proc rw,relatime 0 0
sysfs /sys sysfs rw,seclabel,relatime 0 0
selinuxfs /sys/fs/selinux selinuxfs rw,relatime 0 0
debugfs /sys/kernel/debug debugfs rw,seclabel,relatime 0 0
none /acct cgroup rw,relatime,cpuacct 0 0
none /sys/fs/cgroup tmpfs rw,seclabel,relatime,mode=750,gid=1000 0 0
tmpfs /mnt tmpfs rw,seclabel,relatime,mode=755,gid=1000 0 0
none /dev/cpuctl cgroup rw,relatime,cpu 0 0
/dev/block/mtdblock0 /system ext4 ro,seclabel,relatime,data=ordered 0 0
/dev/block/mtdblock1 /data ext4 rw,seclabel,nosuid,nodev,noatime,nomblk_io_submi
t,data=ordered 0 0
/dev/block/mtdblock2 /cache ext4 rw,seclabel,nosuid,nodev,noatime,data=ordered 0
 0
tmpfs /storage tmpfs rw,seclabel,relatime,mode=755,gid=1000 0 0
/dev/block/vold/public:179,0 /mnt/media_rw/0A19-160E vfat rw,dirsync,nosuid,node
v,noexec,relatime,uid=1023,gid=1023,fmask=0007,dmask=0007,allow_utime=0020,codep
age=cp437,iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
/dev/block/vold/public:179,0 /mnt/secure/asec vfat rw,dirsync,nosuid,nodev,noexe
c,relatime,uid=1023,gid=1023,fmask=0007,dmask=0007,allow_utime=0020,codepage=cp4
37,iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
/dev/fuse /mnt/runtime/default/0A19-160E fuse rw,nosuid,nodev,noexec,noatime,use
r_id=1023,group_id=1023,default_permissions,allow_other 0 0
/dev/fuse /storage/0A19-160E fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,gr
oup_id=1023,default_permissions,allow_other 0 0
/dev/fuse /mnt/runtime/read/0A19-160E fuse rw,nosuid,nodev,noexec,noatime,user_i
d=1023,group_id=1023,default_permissions,allow_other 0 0
/dev/fuse /mnt/runtime/write/0A19-160E fuse rw,nosuid,nodev,noexec,noatime,user_
id=1023,group_id=1023,default_permissions,allow_other 0 0
root@generic:/ #
```

Figura 4.39: Comando que permite a visualização das partições do dispositivo

Na Figura 4.39 percebe-se que as partições acima descritas como as mais relevantes para a análise, encontram-se nos seguintes diretórios: */dev/block/mtdblock1* e */dev/block/vold/public:179,0*. O cartão SD pode ser nomeado de várias maneiras dependendo do dispositivo, neste caso é o */mnt/media_ rw/0A19-160E*. Para capturar as imagens, foi utilizada a ferramenta *dd* e o *netcat*, que auxiliaram na transferência [Lohrum].

A Figura 4.40 mostra a captura da partição */data* e pode-se observar que os dados foram salvos no cartão de memória do dispositivo, enquanto na Figura 4.41 as informações contidas no cartão

de memória, obviamente, não podem ser salvas no mesmo, por isso, o *netcat* foi utilizado.

Ao realizar análises no emulador, o cartão de memória foi utilizado para armazenar o *dump* no caso da partição */data*, porém, no próximo cenário a ser analisado, na Seção 4.4, as transferências serão feitas com o auxílio do *netcat*.

```
root@generic:/sdcard # dd if=/dev/block/mtdblock1 of=/sdcard/data.dd
4194304+0 records in
4194304+0 records out
2147483648 bytes transferred in 1403.914 secs (1529640 bytes/sec)
```

Figura 4.40: Uso do *dd* para capturar a imagem da partição */data*

```
d if=/dev/block/vold/public:179,0 | nc -l 4445
6291456+0 records in
6291456+0 records out
3221225472 bytes transferred in 2323.170 secs (1386564 bytes/sec)
```

Figura 4.41: Uso do *dd* para capturar a imagem do cartão de memória

As Figuras 4.42 e 4.43 mostram as diferenças na transferência do *dump* da memória para o computador do investigador. Na Figura 4.42 foi utilizada uma função do ADB, pois como os dados estavam salvos no cartão de memória, um simples *pull* pôde ser utilizado. Enquanto na Figura 4.43 foi utilizado o *netcat*, os dados não poderiam ser salvos diretamente no dispositivo. O caso da Figura 4.43 é sempre a solução mais adequada para uma melhor preservação e conservação da integridade do dispositivo.

```
raquel@raquel:~$ adb pull /sdcard/data.dd /home/raquel
2009 KB/s (2147483648 bytes in 1043.643s)
```

Figura 4.42: Uso de função do ADB para transferir dados coletados do dispositivo para o computador do investigador

```
raquel@raquel:~$ nc localhost 4445 > sdcard.dd
```

Figura 4.43: Uso do *netcat* para transferir dados coletados do dispositivo para o computador do investigador

Após a captura e transferência dos dados necessários serem concluídos, a análise pode ser feita com o uso do *Autopsy*. Os procedimentos de instalação do TSK e do *Autopsy* estão descritos no Anexo I. A ferramenta deve ser inicializada no terminal do computador do investigador, de acordo com a Figura 4.44.

Após isso, como informado na inicialização, a URL *http://localhost:9999/autopsy* deve ser acessada por meio de um navegador HTML. A janela do terminal não pode ser fechada enquanto a

```
raquel@raquel:~$ sudo autopsy
[sudo] password for raquel:

================================================================================

                    Autopsy Forensic Browser
                http://www.sleuthkit.org/autopsy/
                           ver 2.24

================================================================================
Evidence Locker: /var/lib/autopsy
Start Time: Fri Nov 18 20:05:44 2016
Remote Host: localhost
Local Port: 9999

Open an HTML browser on the remote host and paste this URL in it:

    http://localhost:9999/autopsy

Keep this process running and use <ctrl-c> to exit
```

Figura 4.44: Inicialização da ferramenta *Autopsy*

análise estiver em progresso. A Figura 4.45 mostra o menu principal que é apresentado ao usuário.

Figura 4.45: Menu principal do *Autopsy*

Como visto na Figura 4.45, ao inicializar a ferramenta, pode-se optar por abrir um caso (*Open Case*), criar um novo (*New Case*) ou ainda acessar informações sobre o *Autopsy* em *help*. Na opção *Open Case*, algumas informações podem ser colocadas, como o nome do caso, descrição e nome dos investigadores. Depois será pedido para que um novo *host* seja criado, um mesmo caso pode ter vários *hosts* para serem analisados. Na descrição do *host*, pode-se determinar o nome, descrição, fuso horário (que será importante na criação da *timeline* do caso), um banco de dados específico de *hashes*, que determine quando um certo tipo de arquivo alertar um ser malicioso ou quando o arquivo tiver a suspeita ignorada. As informações apresentadas serão armazenadas no diretório onde a ferramenta foi instalada e, após a criação, o *Autopsy* informará este diretório. Neste cenário específico, o diretório foi o */var/lib/autopsy/Android6_ Analise*.

Depois de adicionar o *host*, deve-se apresentar, então, a imagem a ser analisada e que será diretamente relacionada àquele *host* específico. A Figura 4.46 mostra como se exibe o menu de adição de imagens. O campo *Location* deve ser preenchido com o caminho completo em que a imagem está localizada. Se esta estiver em formato fracionado, ao final do caminho, deve-se suprir

a extensão e acrescentar-se um *. O campo *Type* descreve o tipo da imagem a ser analisada, se é um disco completo de um sistema ou apenas uma partição. E, por fim, o campo *Import Method* vai determinar qual será o método de importação da imagem, pois, para analisá-la, esta deve estar localizada no mesmo diretório em que as informações do *host* foram armazenadas. Assim, ela pode ser importada usando um *link* simbólico, pode ser copiada ou até mesmo movida diretamente para o diretório.

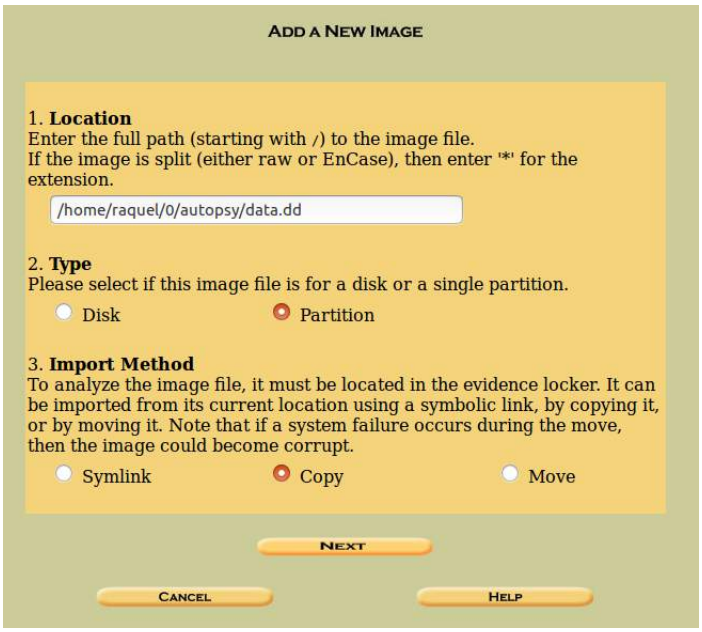


Figura 4.46: Processo de adicionar imagem para análise

Depois da adição, como pode ser visto na Figura 4.47, a ferramenta apresenta alguns detalhes sobre a imagem e permite que a integridade dos dados seja verificada com o cálculo do *hash* MD5. Isto possibilita que seja feita a verificação do *hash* após a importação. Também são especificados detalhes do tipo de sistema de arquivos da imagem. O campo *Mount Point* pode ser mudado para o nome da partição correspondente à imagem. Neste caso, foi deixado o valor padrão */1/*, mas poderia ter sido modificado para */data/*.

Depois que a imagem estiver pronta, a mesma ficará armazenada na galeria de casos e poderá ser analisada. A Figura 4.48 mostra um exemplo em que foram adicionadas não só as partições */data* e a do cartão de memória, mas também a */system* e */cache*. Pode-se perceber também a diferença dos sistemas de arquivo de cada partição, variando entre *ext* e *FAT32*.

Além das informações presentes na Figura 4.48, ao clicar para ver os detalhes da partição, pode-se extrair *strings* de todo o volume e também extrair fragmentos não alocados, que facilitam a busca por palavas-chave e também na recuperação de dados deletados. Estas opções são mostradas na Figura 4.49.

A Figura 4.50 mostra uma visão das análises que podem ser feitas, a partir da imagem escolhida. Na parte superior dela, pode-se visualizar as opções que foram explicadas no Capítulo 3, Seção 3.5:

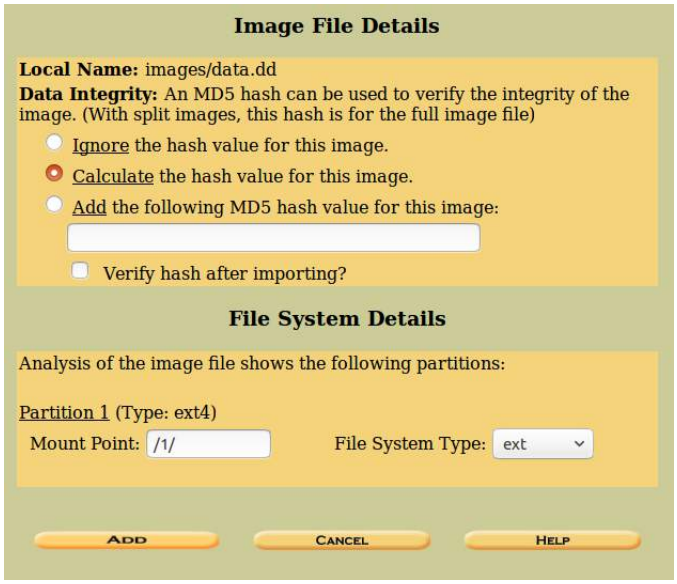


Figura 4.47: Detalhes da imagem adicionada para análise

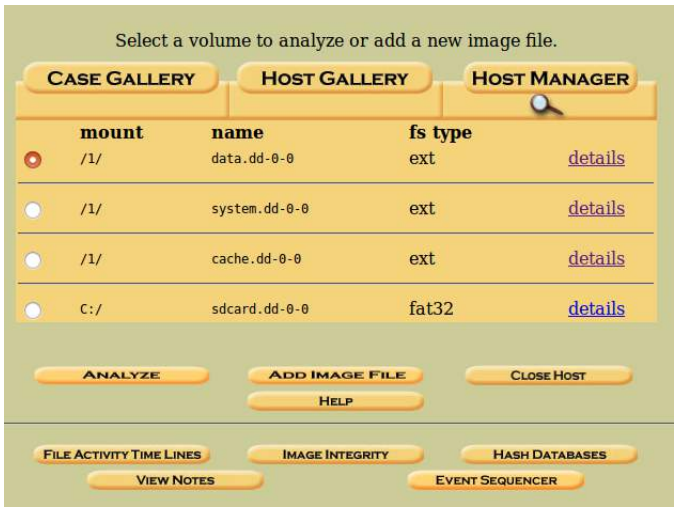


Figura 4.48: Escolha da partição para ser analisada

- 1) **Análise de arquivos (*File Analysis*):** ao selecionar esta opção, todos os diretórios da partição são exibidos para a escolha do investigador;
- 2) **Procura por palavras-chave (*Keyword Search*):** procura arquivos específicos, a partir da entrada de palavras-chave;
- 3) **Tipo de arquivo (*File Type*):** opção em que é possível ordenar os arquivos em categorias, de acordo com suas extensões;
- 4) **Detalhes da imagem (*Image Details*):** detalhes acerca da imagem em exame;
- 5) **Metadados (*Meta Data*):** visualização das informações, a partir do *inode* do arquivo;

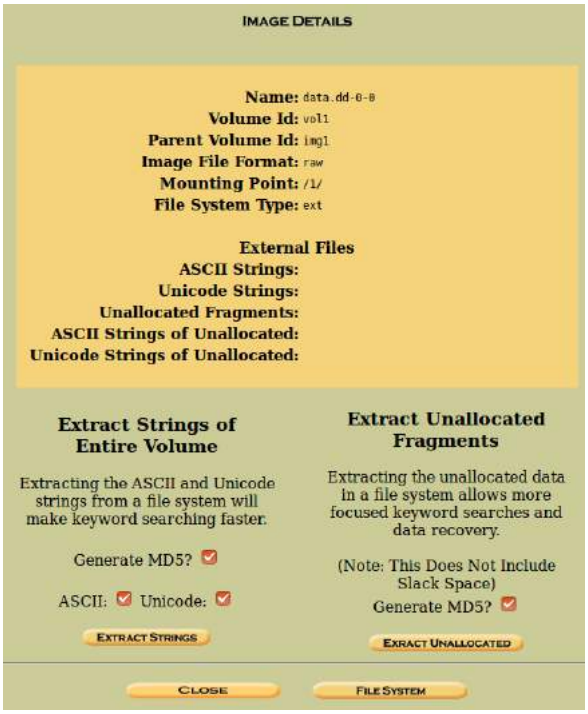


Figura 4.49: Opções adicionais para as partições analisadas

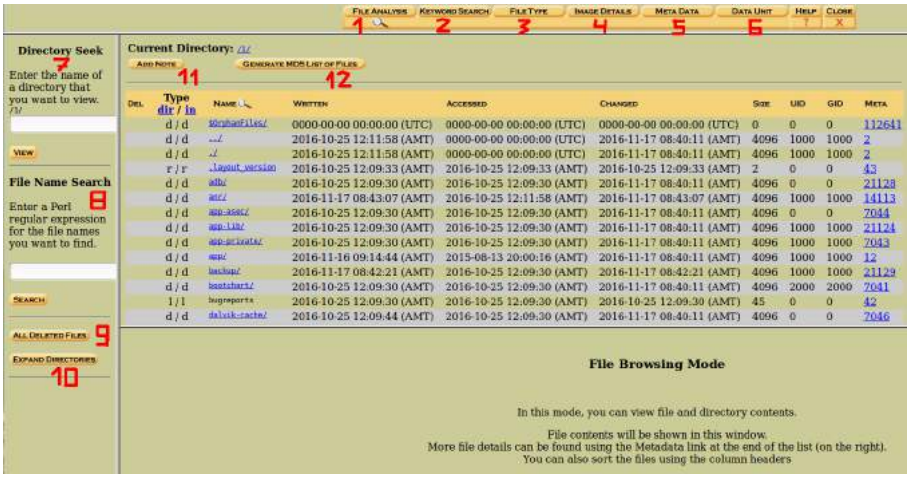


Figura 4.50: Visão geral das análises que podem ser feitas com a imagem selecionada

- 6) **Unidade de dados (*Data Unit*):** visualização do conteúdo de qualquer fragmento de determinados arquivos;
- 7) **Procura de diretórios (*Directory Seek*):** procura por diretórios específicos;
- 8) **Nome de arquivos (*File Name*):** procura por nomes de arquivos específicos;
- 9) **Todos os arquivos deletados (*All Deleted Files*):** mostra os arquivos deletados reconhecidos pela ferramenta;

- 10) **Expansão dos diretórios (*Expand Directories*):** expande os diretórios para que sejam exibidos de maneira completa;
- 11) **Acréscimo de notas (*Add Note*):** função que permite que sejam adicionadas notas com observações sobre certos arquivos;
- 12) **Criação do MD5 de uma lista de arquivos (*Generate MD5 List of Files*):** gera uma lista com o *hash* MD5 dos arquivos.

A seguir, serão apresentados os resultados de algumas destas opções para as partições */data* e do cartão de memória.

4.3.1 Análise da partição */data*

Para organizar a busca de dados, o primeiro passo que pode ser feito é a ordenação dos arquivos, de acordo com seus tipos. Assim, o investigador pode procurar por aquilo que achar mais útil. Para isso, é preciso ir até a aba *File Type*, opção 3 da Figura 4.50. Ao acessá-la, num menu à esquerda da tela, terá duas alternativas, a que deve-se escolher é a *Sort Files by Type*. A Figura 4.51 mostra o resultado desta operação.

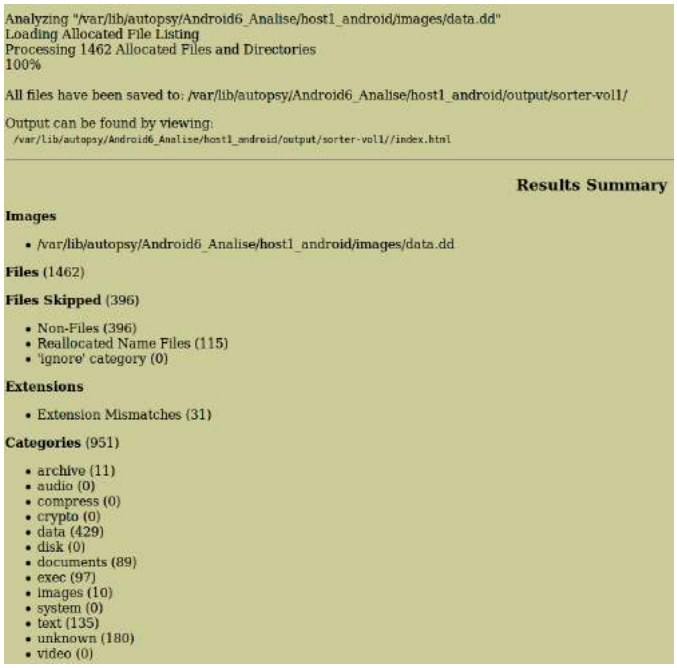


Figura 4.51: Resultado da organização dos dados de acordo com suas extensões

Após isso, como informado na Figura 4.51, os dados organizados estarão disponíveis para visualização no mesmo local, onde a ferramenta foi instalada, em uma pasta específica. Neste caso, foi a */var/lib/autopsy/Android6_Analise/host1_android/output/sorter-vol1/index.html*. Ao acessar o arquivo *index.html*, haverá o item *categories* em que pode-se ter acesso a cada tipo de dados específicos. Por exemplo, ao acessar a categoria *images*, outro arquivo, o *images.html*, é

aberto e, então, pode-se visualizar todos os diretórios do dispositivo em que foram encontrados arquivos, considerados imagens. A Figura 4.52 mostra como isso é determinado.

```
/1/backup/com.android.internal.backup.LocalTransport/com.android.providers.settings
  X11 SNF font data, MSB first
  Image: /var/lib/autopsy/Android6_Analise/host1_android/images/data.dd Inode: 21339

/1/system/recent_images/8_task_thumbnail.png
  PNG image data, 576 x 576, 8-bit/color RGBA, non-interlaced
  Image: /var/lib/autopsy/Android6_Analise/host1_android/images/data.dd Inode: 21335

/1/system/recent_images/6_task_thumbnail.png
  PNG image data, 576 x 576, 8-bit/color RGBA, non-interlaced
  Image: /var/lib/autopsy/Android6_Analise/host1_android/images/data.dd Inode: 21208
```

Figura 4.52: Resultado da filtragem por tipo de arquivos, neste caso as imagens

Com as informações de onde as imagens foram encontradas, pode-se procurar por cada uma delas na opção 7 da Figura 4.50, *Directory Seek*. Nesse caso, a maioria das imagens encontrava-se no diretório */system/recent_ images*. Neste mostra as imagens mais recentes do dispositivo, ou seja, as últimas ações que o usuário realizou. A Figura 4.53 mostra um dos resultados encontrados e percebe-se que se trata da agenda de contatos do dispositivo, o que pode ser importante para uma investigação. Como pode ser visto, o *Autopsy* mostra a data em que a ação foi realizada, o que ajuda na criação de uma *timeline* do caso.

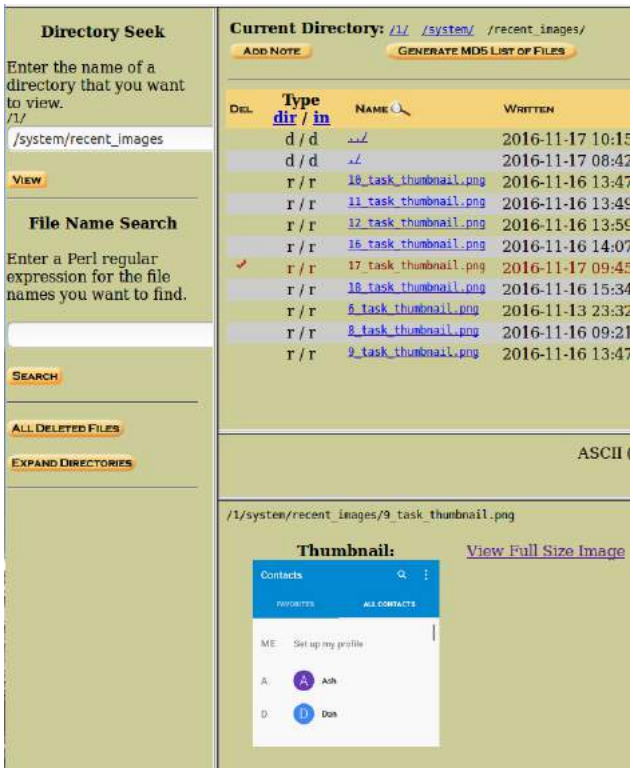


Figura 4.53: Imagem encontrada com o auxílio da filtragem de tipos de arquivos

Além de imagens, os resultados que são de grande utilidade em uma investigação de um dis-

positivo móvel, são os contatos do celular, registros de chamadas, informações sobre os SMS's e MMS's enviados e recebidos, possíveis e-mails de diferentes contas, históricos de URL's acessadas pelo usuário em diferentes navegadores, informações de aplicativos como *Facebook*, *Twitter*, *Snapchat*, *Skype*, entre outros, também podem ser de grande valor, dependendo do suspeito e tipo de caso.

Além disso, também pode-se obter informações acerca das permissões dos aplicativos instalados no dispositivo ao se fazer uma análise à procura de possíveis *malwares*. Essas informações podem ser coletadas e analisadas a partir da imagem da partição */data*, pois são armazenadas na mesma. A seguir, algumas destas análises serão discutidas, outras serão mais detalhadas posteriormente na Seção 4.4.

Para encontrar as outras informações importantes, é essencial analisar tudo que está armazenado no diretório */data/data/*, em que estão contidos os bancos de dados e estes, em sub-diretórios desta partição. O processo de captura deles está descrito nos itens a seguir:

- **Contatos:** a agenda de contatos de um dispositivo tem grande valor para uma investigação, através do *Autopsy* pode-se obter estas informações acessando o seguinte diretório: */data/data/com.android.providers.contacts/databases/*. Neste diretório específico, fica armazenado o arquivo *contacts2.db* que guarda as informações sobre os contatos do dispositivo. Para uma melhor análise, pode ser feita a exportação do banco de dados e, com o auxílio de um editor de arquivos SQL, visualizar em formato de tabelas as informações. Nas análises do trabalho, foi utilizado o *DM Browser for SQLite* [SQLite] para visualizar as informações. A Figura 4.54 mostra os resultados obtidos.

data1	data2
Filter	Filter
012-9	2
Ash	Ash
ash@em.com	1
1 234-56	2
Dan	Dan
dan@em.com	1

Figura 4.54: Informações de contatos presentes no dispositivo

- **Chamadas:** as chamadas efetuadas, recebidas, perdidas, são de extrema importância em uma investigação. E estes dados podem ser encontrados no mesmo arquivo de banco de dados dos contatos. Da mesma maneira que nos contatos, foi visualizado o arquivo *contacts2.db* e os resultados de chamadas estão exemplificados na Figura 4.55.

- **Informações sobre operadoras utilizadas:** este tipo de informação também pode ser útil e a mesma é localizada na partição */data/data/com.android.providers.telephony/databases/*, no arquivo *telephony.db*. Utilizando o editor de arquivos SQL, a Figura 4.56 demonstra os

number	presentation	date	duration	data_usage	type
Filter	Filter	Filter	Filter	Filter	Filter
123456	1	147931830...	12	NULL	1
0129	1	147931834...	0	NULL	3

Figura 4.55: Informações de chamadas presentes no dispositivo

resultados das informações. Como optou-se pelo emulador, as informações serão genéricas, na Seção 4.4 estas trarão mais detalhes.

_id	name	numeric	mcc	mnc	apn
Filter	Filter	Filter	Filter	Filter	Filter
1	T-Mobile US	310260	310	260	epc.tmobile.com
2	T-Mobile US 250	310250	310	250	epc.tmobile.com
3	T-Mobile US 660	310660	310	660	epc.tmobile.com
4	T-Mobile US 230	310230	310	230	epc.tmobile.com
5	T-Mobile US 310	310310	310	310	epc.tmobile.com

Figura 4.56: Informações de operadoras presentes no dispositivo

- **SMS's e MMS's:** também de extrema importância, estão localizadas na mesma partição das informações sobre operadoras de telefonia, porém, no arquivo *mmssms.db*. Neste caso, o emulador não possuía nenhuma das duas informações para demonstração. Estes dados serão coletados novamente na Seção 4.4.

Como discutido, informações acerca de aplicativos específicos serão lidadas na Seção 4.4, em que se analisa um estudo de caso utilizando não mais emuladores e sim um dispositivo móvel real.

4.3.2 Análise da partição do cartão de memória

As principais informações que podem ser extraídas de um cartão de memória de um dispositivo, são as fotos da câmera (que, na maioria das vezes, são salvas neste dispositivos externos), documentos, e até mesmo algumas informações de aplicativos que possam ter sido salvas no cartão de memória. Neste caso, não foram instalados aplicativos no cartão de memória do emulador, porém, existiam imagens e outros documentos salvos no mesmo.

A Figura 4.57 mostra a visão geral da imagem do cartão de memória do dispositivo. Pode-se perceber que existem algumas pastas que levam a outros diretórios e também outros documentos como *config* e *mobile* que estão no diretório principal do cartão SD. Para se saber a procedência, formato e outras informações sobre cada um destes arquivos, deve-se clicar sobre eles e, então, o tipo de arquivo será informado pelo *Autopsy*.

Ainda na Figura 4.57 pode-se confirmar esse processo, já que ao clicar sobre o arquivo *mobile*, o *Autopsy* informa que é um documento no formato PDF, de versão 1.5. Para se comprovar, basta

exportar o arquivo para o computador do investigador e então executá-lo.

Type dir / in	Name	Written	Accessed	Created	Size
v / v	$FAT1	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	6279168
v / v	$FAT2	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	6279168
v / v	$MBR	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	512
d / d	$OrphanFiles/	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0
d / d	.android_secure/	2016-11-14 01:01:02 (AMT)	2016-11-14 00:00:00 (AMT)	2016-11-14 01:01:02 (AMT)	2048
d / d	Android/	2016-11-14 05:09:30 (AMT)	2016-11-14 00:00:00 (AMT)	2016-11-14 05:09:30 (AMT)	2048
r / r	cache.dd	2016-11-17 13:30:54 (AMT)	2016-11-17 00:00:00 (AMT)	2016-11-17 13:30:54 (AMT)	69206016
r / r	config	2016-10-17 15:08:46 (AMT)	2016-10-17 00:00:00 (AMT)	2016-11-16 15:49:14 (AMT)	28291
d / d	DCIM/	2016-11-17 15:27:10 (AMT)	2016-11-17 00:00:00 (AMT)	2016-11-17 15:27:10 (AMT)	2048
r / r	lime.ko	2016-11-16 11:36:06 (AMT)	2016-11-16 00:00:00 (AMT)	2016-11-16 16:05:38 (AMT)	7004
d / d	LOST.DIR/	2016-11-14 01:01:02 (AMT)	2016-11-14 00:00:00 (AMT)	2016-11-14 01:01:02 (AMT)	2048
r / r	mobile	2016-11-09 14:30:30 (AMT)	2016-11-09 00:00:00 (AMT)	2016-11-16 15:48:46 (AMT)	889530
r / r	teste.dd	2016-11-17 11:42:34 (AMT)	2016-11-17 00:00:00 (AMT)	2016-11-17 11:42:34 (AMT)	7458816

ASCII (display - report) * Hex (display - report) * ASCII Strings (display - report)
File Type: PDF document, version 1.5

Figura 4.57: Visão geral do cartão de memória e identificação do formato do arquivo *mobile*

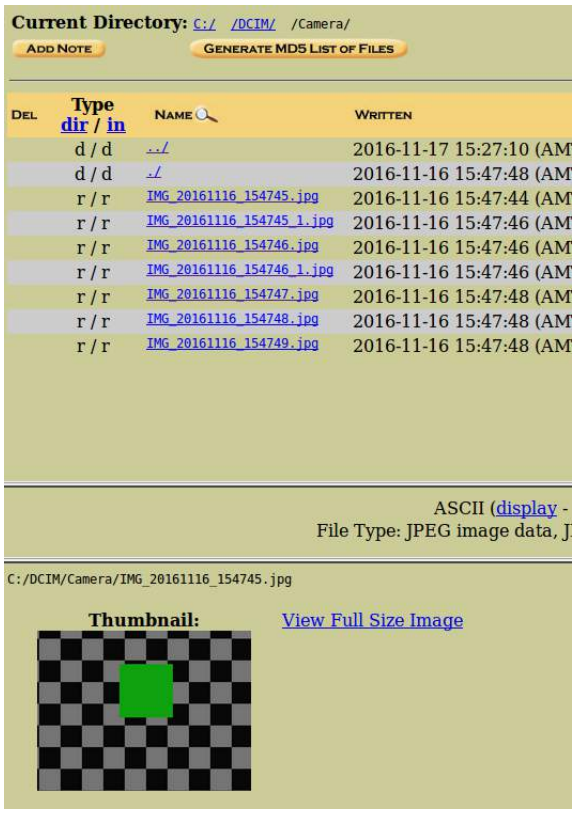


Figura 4.58: Exemplo de uma imagem recuperada do cartão de memória do dispositivo

Um outro detalhe a ser observado na Figura 4.57, é que existe um arquivo chamado *teste.dd*, que está em vermelho. Quando isto acontece, significa que este arquivo foi deletado do dispositivo, mas o *Autopsy* foi capaz de recuperá-lo. Algumas vezes, é possível que este seja 100% recuperado e, portanto, pode ser executado para ter seu conteúdo examinado; outras vezes, a ferramenta só

consegue expor o arquivo, porém, não foi capaz de resgatar suas informações completas. Nestes casos de não recuperação, pode-se usar a técnica de *carving* de arquivos, esta será detalhada na Seção 4.4.

Os outros dados mais importantes a serem extraídos, são as imagens. Estas são armazenadas na pasta *DCIM/* e, ao acessá-la, estarão em *Camera/* e também terão suas miniaturas salvas em *.thumbnails/*. A Figura 4.58 mostra o resultado desta análise.

4.4 Terceiro cenário - Samsumg, modelo Galaxy S4, GT - I9515L, Versão do Android 5.0.1

Nesta seção, serão detalhados três casos, descritos na Tabela 4.1, que o analista pericial pode encontrar. Para cada um deles será considerado como objetivo, a extração das informações que o analista julgar como relevante, entre elas, estão agenda de contatos, registros de chamadas, SMS's, e-mails, imagens, vídeos, arquivos, informações em Redes Sociais, entre outros. Também serão utilizadas algumas das ferramentas que foram detalhadas, sendo assim, só serão expostos os principais resultados e não mais todo o passo a passo de instalação e uso. O *smartphone* utilizado em todos os casos foi o mesmo, o Samsumg, modelo Galaxy S4, GT - I9515L, Android Versão 5.0.1.

Tabela 4.1: Descrição dos casos que serão analisados

Características do *Smartphone*				
Casos	Ligado	Bloqueado	Acesso ADB	*Root*
Caso 1	Sim	Não	Não	Não
Caso 2	Sim	Sim	Não	Não
Caso 3	Sim	Sim	Sim	Sim

4.4.1 Caso 1 - Dispositivo ligado, sem bloqueio de tela, sem acesso ADB e sem permissões de *root*

Neste primeiro caso, considera-se que o dispositivo em análise esteja ligado, sem nenhum tipo de bloqueio de tela, sem acesso ADB e sem permissões de *root*. O primeiro passo tomado foi ativar o modo de desenvolvedor no dispositivo, para que então o acesso ADB fosse ativado, a fim de que o dispositivo pudesse comunicar-se via USB com o computador.

Para ativar o modo de desenvolvedor, foi necessário acessar a opção Configurações. Ao processá-la, o menu *MAIS* teve que ser selecionado. Já nele, o próximo passo, foi clicar em *Sobre o dispositivo* (alguns aparelhos celulares já possuem a opção *Sobre o dispositivo* diretamente no menu Configurações), descrito na Figura 4.59. Neste sub-menu, existe a opção *Número de compilação* (ou *Build number/Baseband version* em outros telefones) (Figura 4.59), esta teve que ser selecionada por 7 vezes consecutivas, para que então o analista pudesse se tornar desenvolvedor, como descrito na Figura 4.60. Após este procedimento, no menu *MAIS*, a *tag Opções do desenvolvedor* foi ativada

(Figura 4.60). Ao selecioná-la a *Depuração de USB* (ou *USB debugging* em outros dispositivos) pôde ser ativada, como está descrito na Figura 4.60.

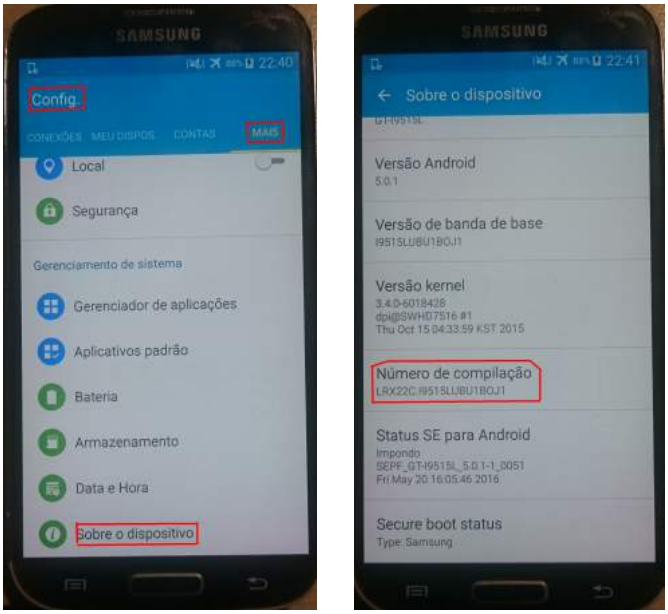


Figura 4.59: Opções que devem ser acessadas no menu Configurações

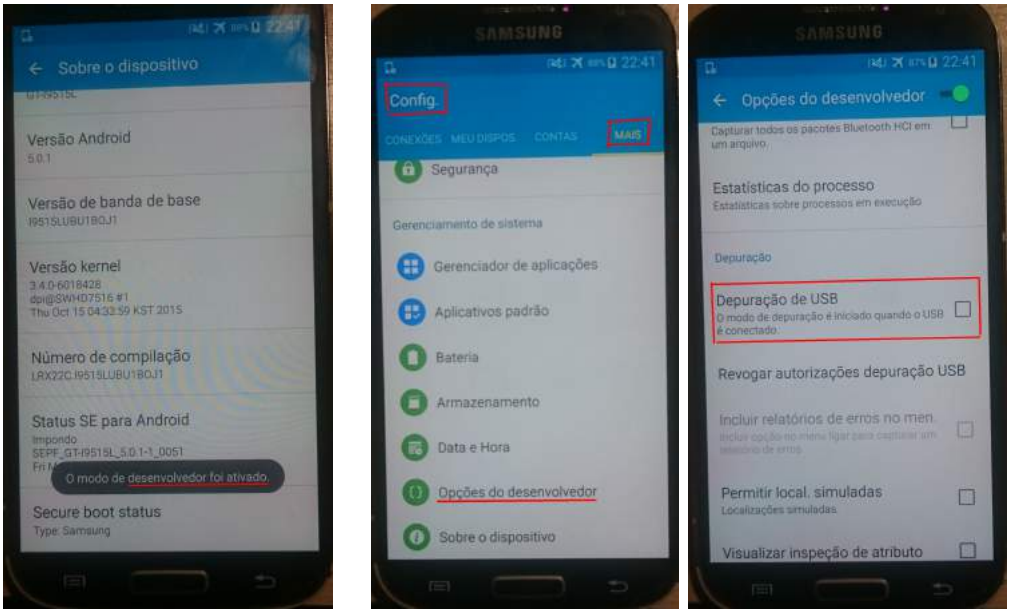


Figura 4.60: Opções de desenvolver ativada

Porém, além de ativar a opção de desenvolvedor, precisou-se de permissões de super usuário para efetuar as operações necessárias, a fim de obter os dados do celular com as ferramentas apresentadas. Por isso, foi preciso efetuar alguns procedimentos para obter *root* no dispositivo. Estes serão apresentados a seguir.

Foi feito o *download* da nova imagem ROM, carregada na memória do dispositivo. A imagem

utilizada foi a presente no endereço eletrônico citada na nota [CF-Auto-Root]. Depois de feito o *download*, o arquivo estava já compactado e precisou ser descompactado. Depois de ter sido descompactado, foi encontrado um arquivo da ferramenta *Odin* (que só possui suporte para Windows e, portanto, não foi utilizada) e também o arquivo *CF-Auto-Root-jfvelte-jfvelteub-gti9515l.tar.md5*. Este último arquivo foi também descompactado (antes excluiu-se o termo *.md5*) para que fossem obtidos dois arquivos, o *recovery.img* e o *cache.img.ext4* que substituiriam, posteriormente, os arquivos originais do dispositivo. Para a transferências destes arquivos, foi utilizada a ferramenta *Heimdall*, cujo procedimento de instalação está descrito no Anexo I.

Com o dispositivo desligado, conectou-se o mesmo ao computador via cabo USB. Foi preciso que o mesmo fosse colocado em modo *debug*. Isto é feito apertando-se ao mesmo tempo três teclas do dispositivo: volume para baixo + botão central abaixo da tela + botão de ligar. Aparecerá, então, uma tela alertando os perigos de realizar um procedimento deste tipo no aparelho, ilustrado na primeira imagem da Figura 4.61, cuja mensagem deve ser ignorada e, para continuar, deve-se apertar a tecla de volume para cima. Outra mensagem surgirá, desta vez de que o dispositivo está pronto para *download*, como demonstra a Figura 4.61. Depois desta nova mensagem, no computador, abre-se um terminal e no diretório onde se encontram os arquivos que foram descompactados acima, se faz o procedimento descrito na Figura 4.62.

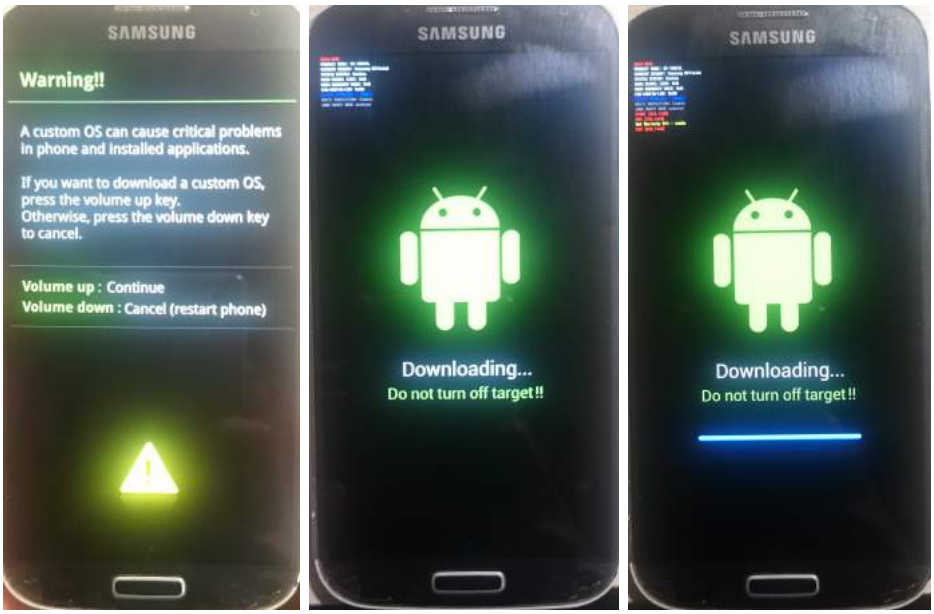


Figura 4.61: Instruções do procedimento de instalação da nova ROM

Se todo o procedimento estiver funcionando, uma barra de progresso azul aparecerá no dispositivo como está mostrado na terceira imagem da Figura 4.61. Depois que o procedimento estiver finalizado, religue o celular e alguns passos serão feitos pelo próprio dispositivo. Se esses não forem feitos, coloque o aparelho em modo *recovery*, apertando as teclas volume para cima + botão central + botão de ligar, ao mesmo tempo. A logo da Samsung irá aparecer com algumas mensagens no canto superior esquerdo da tela, depois disso, o celular inicializará normalmente e algumas mensagens de atualização do Android aparecerão no dispositivo. Pronto! O dispositivo já está em modo

```
raquel@raquel:~/Documents/CF-Auto-Root-jfvelte-jfvelteub-gti9515l$ sudo heimdal
l flash --RECOVERY recovery.img --CACHE cache.img.ext4 --no-reboot
Heimdall v1.4.1

Copyright (c) 2010-2014 Benjamin Dobell, Glass Echidna
http://www.glassechidna.com.au/

This software is provided free of charge. Copying and redistribution is
encouraged.

If you appreciate this software and you would like to support future
development please consider donating:
http://www.glassechidna.com.au/donate/

Initialising connection...
Detecting device...
Claiming interface...
Setting up interface...

Initialising protocol...
Protocol initialisation successful.

Beginning session...

Some devices may take up to 2 minutes to respond.
Please be patient!

Session begun.

Downloading device's PIT file...
PIT file download successful.

Uploading RECOVERY
100%
RECOVERY upload successful

Uploading CACHE
100%
CACHE upload successful

Ending session...
Releasing device interface...
```

Figura 4.62: Procedimento para instalação dos novos arquivos no dispositivo

root.

Depois que os procedimentos de ativação das opções de desenvolvedor e de configuração de super usuário foram finalizadas, pôde-se, então, efetuar os mesmos procedimentos da Seção 4.3 analisando-se a memória persistente do dispositivo.

Para começar as análises, a imagem das principais partições do dispositivo, tiveram que ser feitas. A Figura 4.63 mostra as partições do mesmo. Nesta análise, as partições foram copiadas diretamente para o computador do investigador e, portanto, usou-se o *netcat*. Para utilizá-lo, a ferramenta *busybox* foi instalada no dispositivo, cujo procedimento está no Anexo I. Após isso, uma conexão TCP foi iniciada com o ADB para que a transferência fosse feita. E, então, a partição foi copiada de acordo com a Figura 4.64. Neste cenário, o *smartphone* não possuía cartão de memória, e somente a partição */data* foi analisada.

Depois deste procedimento, foi aberto um novo caso no *Autopsy* para que a imagem fosse analisada. Cada sub-seção que será tratada, apresentará diferentes resultados, a partir da memória persistente analisada.

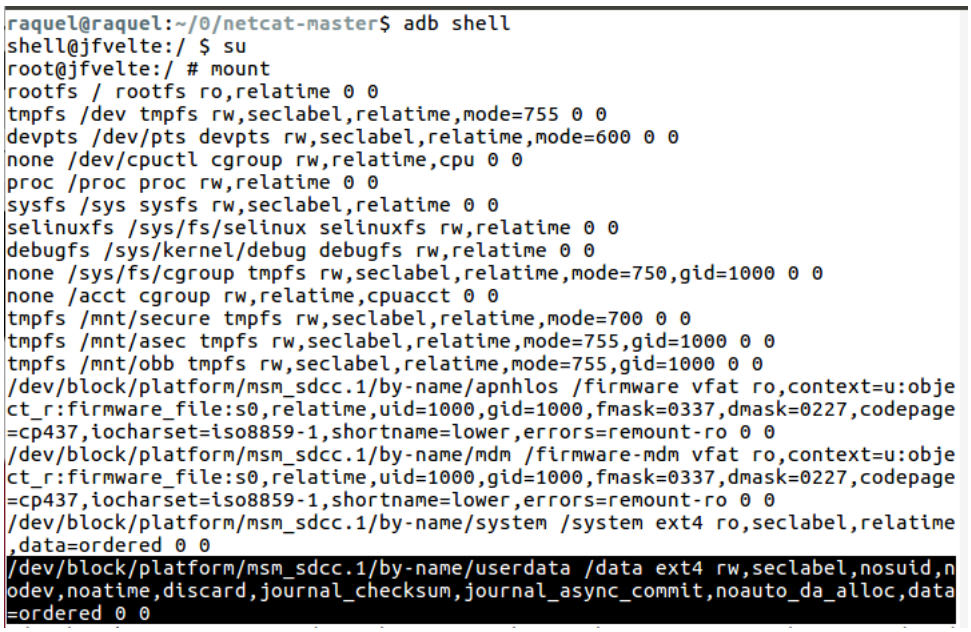


Figura 4.63: Partições do dispositivo

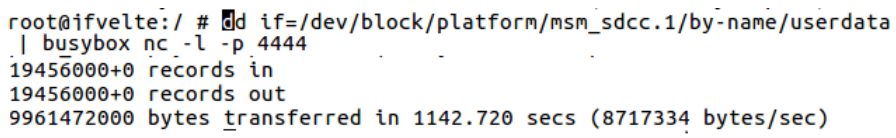


Figura 4.64: Processo de imagem da partição */data*

4.4.1.1 Agenda de Contatos e Registros de Chamadas

As primeiras informações que usualmente são extraídas de um dispositivo, são a agenda de contatos e registros de chamadas. Tanto os contatos quanto os registros de chamadas são armazenados no mesmo banco de dados. Os contatos não são necessariamente salvos pelo usuário, ao enviar um e-mail por meio do aplicativo *Gmail*, o contato pode ser salvo automaticamente pelo próprio *Gmail* na agenda de contatos por exemplo.

O nome do pacote que armazena as informações referentes aos contatos e chamadas, como foi visto na Seção 4.3, é o *com.android.providers.contacts*. Ao analisar a imagem no *Autopsy*, pode-se procurar diretamente pelo nome do pacote. A Figura 4.65 mostra que a opção *File Analysis* deve ser selecionada e então o pacote alvo pode ser procurado tanto na opção *Directory Seek* como em *File Name Search*. No caso da pesquisa em *Directory Seek*, deve-se inserir o caminho completo do pacote, isto é, adicionar a partição à qual ele pertence que, neste caso, e nas próximas análises, é a */data/data/* [Tamma e Tindall 2015].

Como pode ser visto na Figura 4.65, existem vários sub-diretórios para serem analisados. Os mais importantes, neste caso, são: */files/photos* e */files/profiles* onde serão armazenadas, respectivamente, fotos dos contatos do dispositivo e a foto de perfil do usuário. A outra partição importante

Figura 4.65: Pacote em que são armazenadas informações referentes aos Contatos e Chamadas

é a */databases/*, que contém vários arquivos de banco de dados, um deles é o *contacts2.db*, que armazena as informação sobre chamadas feitas do e para o dispositivo e também todos os contatos.

No aparelho analisado, o usuário não atribuiu foto alguma aos seus contatos e nem possuía uma foto de perfil. Para analisar, então, o principal banco de dados, *contacts2.db*, é preciso acessar o diretório informado acima e, para uma melhor visualização, o arquivo pôde ser exportado para que pudesse ser analisado em um editor e visualizador de arquivos SQL. Usou-se a mesma ferramenta da Seção 4.3, *DB Browser for SQLite* (instruções de instalação no Anexo I) para a análise.

Nos arquivos *database* do Android, as informações são divididas por tabelas e cada uma delas traz um tipo de dado diferente. No caso dos contatos são divididas nas seguintes tabelas [Tamma e Tindall 2015]:

- ***accounts*:** mostra as contas registradas no dispositivo que têm acesso à lista de contatos. Ao menos uma das contas mostrará o endereço de e-mail da conta do Google do usuário. A lista também pode incluir *apps* não desenvolvidos pelo Google, mas que possuem permissão para acessar a lista de contatos.
- ***calls*:** contêm informações com relação às chamadas do dispositivo. Possui algumas colunas específicas com informações importantes:
  - ***Number*:** mostra o número de telefone do usuário remoto, independentemente se a chamada foi efetuada ou recebida;
  - ***Date*:** mostra o dia e a hora da chamada e é armazenada no formato *Epoch* do Linux;
  - ***Duration*:** duração da chamada, em segundos;
  - ***Type*:** indica o tipo de ligação, se o valor foi '1', significa que foi uma chamada recebida, '2' significa que a chamada foi efetuada pelo dispositivo e '3' que a chamada foi perdida;
  - ***Name*:** nome do contato correspondente à chamada, se o mesmo estiver armazenado na lista de contatos;

  - ***Geocoded_location*:** mostra a localização do número de telefone, de acordo com o código de área ou código do país.

- ***contacts*:** contêm informações parciais sobre os contatos, também possui colunas específicas:
  - ***name_raw_contact_id*:** seu valor corresponde aos parâmetros *_id value* da tabela *raw_contacts* (que será apresentada posteriormente);
  - ***photo_file_id*:** corresponde ao nome do arquivo encontrado no diretório */files/photos*, em que são armazenadas as fotos da lista de contatos;
  - ***times_contacted*** e ***last_times_contacted*:** mostram, respectivamente, o número de vezes que o contato ligou para o dispositivo ou que foram feitas ligações para o mesmo e o dia e a hora em que a última ligação relacionada a um contato foi feita (no formato *Epoch* do Linux).

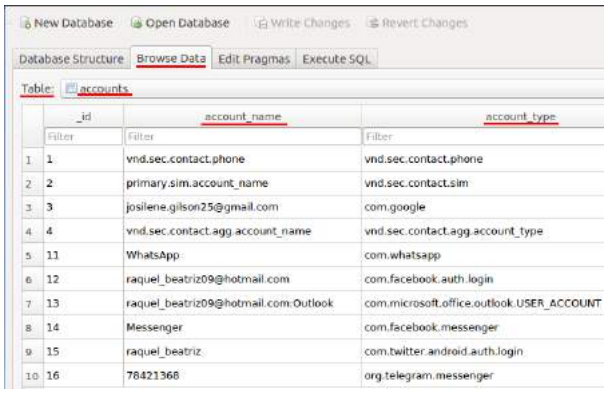
- ***data*:** contém as informações de cada contato: números de telefone, endereços de e-mail, e assim por diante. Possui as seguintes colunas importantes:
  - ***raw_contact_id*:** é um valor único para cada contato que pode servir para identificar o mesmo;
  - ***data1...data15*:** contém informações sobre o contato, mas não existe um padrão perceptível. A mesma coluna *data* pode conter o nome do contato, um endereço de e-mail, um perfil de uma Rede Social, entre outros. A coluna *data14* é reservada para se relacionar com os nomes de arquivos das imagens de perfil de um contato. E a coluna *data15* contém um *thumbnail* da foto de perfil de um contato.

- ***deleted_contacts*:** contêm uma coluna *contact_id* que é a identificação do contato apagado e a coluna *deleted_contact_timestamp* que mostra no format *Epoch* o dia e hora em que foi deletado. Apesar destas informações, não é possível relacionar estes valores com as outras tabelas, para recuperar os dados é preciso aplicar técnicas de recuperação de arquivos deletados;

- ***groups*:** mostra grupos criados na lista de contatos, que podem ser gerados automaticamente ou criados pelo usuário. A coluna *title* mostra o nome dos grupos, porém, não é possível identificar os usuários presentes em cada grupo;

- ***raw_contacts*:** contém as informações dos contatos de forma mais compacta que a tabela *data*. As seguintes colunas podem ser destacadas:
  - ***display_name*:** mostra o nome do contato, se este estiver disponível. Para determinar o número de telefone, endereço de e-mail ou outras informações do contato, o valor da coluna *_id* tem que coincidir com o valor da coluna *raw_contact_id* da tabela *data*;
  - ***times_contacted*** e ***last_times_contacted*:** são os mesmos dados exibidos pela mesma coluna na tabela *contacts* e se aplica somente para ligações telefônicas, o envio de um e-mail ou SMS para um contato não incrementa este valor.

A Figura 4.66 mostra com o auxílio da ferramenta que pode ser encontrada no endereço eletrônico [SQLite], como buscar pelas tabelas de uma maneira mais simples. Pode-se perceber que, neste caso, está sendo visualizada a tabela *accounts*, mostrando portanto, as contas que estão registradas no dispositivo, inclusive de aplicativos não desenvolvidos pelo Google, como as contas do *Twitter*, *Facebook* e *Outlook*.



	_id	account_name	account_type
	Filter	Filter	Filter
1	1	vnd.sec.contact.phone	vnd.sec.contact.phone
2	2	primary.sim.account_name	vnd.sec.contact.sim
3	3	josilene.gilson25@gmail.com	com.google
4	4	vnd.sec.contact.agg.account_name	vnd.sec.contact.agg.account_type
5	11	WhatsApp	com.whatsapp
6	12	raquel_beatriz09@hotmail.com	com.facebook.auth.login
7	13	raquel_beatriz09@hotmail.com:Outlook	com.microsoft.office.outlook.USER_ACCOUNT
8	14	Messenger	com.facebook.messenger
9	15	raquel_beatriz	com.twitter.android.auth.login
10	16	78421368	org.telegram.messenger

Figura 4.66: Pacote em que são armazenadas informações referentes aos Contatos e Chamadas

Table: contacts

	_id	name_raw_contact_id	times_contacted	last_time_contacted
	Filter	Filter	Filter	Filter
1	12908	12897	334	1478252706313
2	9620	9486	297	1478255169191
3	12933	12799	287	1478253419034
4	13027	12120	272	1478185766170
5	12849	12744	260	1478205477430
6	12864	12730	258	1478209378769
7	12940	12805	239	1478098683125
8	12882	12737	226	1478255154159
9	9653	9519	189	1477613875375
10	13028	12490	148	1478111402846

Figura 4.67: Tabela *contacts* onde pode-se visualizar para quais contatos foram feitas mais ligações

Na Figura 4.67 pode-se observar a tabela *contacts*. Para que os contatos que estão sendo informados na coluna *name_ raw_ contact_ id* sejam descobertos, observa-se a tabela *raw_ contacts* para que a informação de quantas vezes um contato específico tenha sido contactado seja completa.

Ao analisar a tabela *raw_ contacts*, foi constatado que o contato que tinha como *name_ raw_ contact_ id* o número '12897' e que foi contactado 334 vezes pelo usuário do dispositivo, estava registrado na lista de contatos como 'Josilene', conforme descrito na Figura 4.68. A partir desta informação e, dependendo do caso e foco de investigação, pode-se analisar a relação do proprietário do aparelho com estes contatos.

A tabela *data* apresenta detalhes sobre os contatos, com seus números de telefone e informações adicionais. Além disso, também são registrados os e-mails que foram usados em aplicativos. Estas informações são demonstradas nas Figuras 4.69 e 4.70.

Table: raw_contacts

	_id	display_name	display_name_alt
	Filter	Filter	Filter
1	12908	[redacted]	[redacted]
2	12907	[redacted]	[redacted]
3	12906	[redacted]	[redacted]
4	12905	[redacted]	[redacted]
5	12904	[redacted]	[redacted]
6	12903	[redacted]	[redacted]
7	12902	[redacted]	[redacted]
8	12901	[redacted]	[redacted]
9	12900	[redacted]	[redacted]
10	12899	[redacted]	[redacted]
11	12898	[redacted]	[redacted]
12	12897	Josilene	Josilene

Figura 4.68: Tabela *raw_ contacts* com informação que identifica o contato mais frequente nas ligações

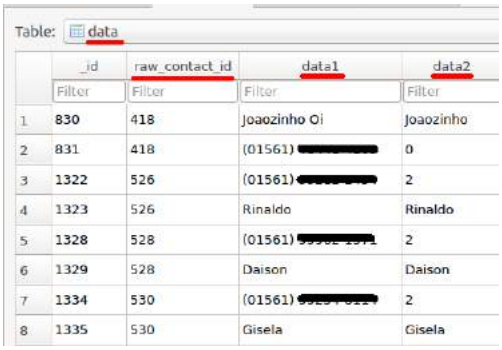


Table: data

	_id	raw_contact_id	data1	data2
	Filter	Filter	Filter	Filter
1	830	418	Joaozinho Oi	Joaozinho
2	831	418	(01561) [redacted]	0
3	1322	526	(01561) [redacted]	2
4	1323	526	Rinaldo	Rinaldo
5	1328	528	(01561) [redacted]	2
6	1329	528	Daison	Daison
7	1334	530	(01561) [redacted]	2
8	1335	530	Gisela	Gisela

Figura 4.69: Informações dos números dos contatos da tabela *data*

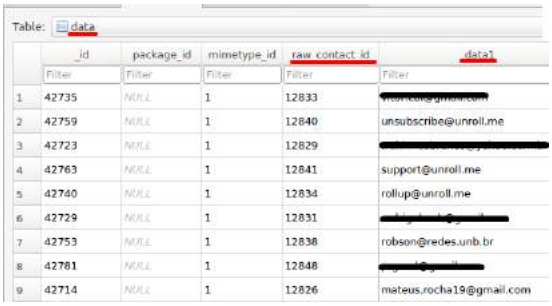


Table: data

	_id	package_id	mimetype_id	raw_contact_id	data1
	Filter	Filter	Filter	Filter	Filter
1	42735	NULL	1	12833	[redacted]
2	42759	NULL	1	12840	unsubscribe@unroll.me
3	42723	NULL	1	12829	[redacted]
4	42763	NULL	1	12841	support@unroll.me
5	42740	NULL	1	12834	rollup@unroll.me
6	42729	NULL	1	12831	[redacted]
7	42753	NULL	1	12838	robson@redes.unb.br
8	42781	NULL	1	12848	[redacted]
9	42714	NULL	1	12826	mateus.rocha19@gmail.com

Figura 4.70: Informações de endereços de e-mail registrados no dispositivo da tabela *data*

4.4.1.2 Análises de SMS's/MMS's

Outro conjunto de informações essenciais, são os SMS's que possam ter sido enviados e/ou recebidos. Além disso, as mensagens multimídias – MMS existentes nos dispositivos, também podem conter dados importantes. Tanto os SMS's quanto MMS's são armazenados no mesmo banco de dados. O nome do pacote que armazena as informações referentes a estes dados, é

o *com.android.providers.telephony* [Tamma e Tindall 2015]. Foi feito o mesmo procedimento da seção anterior em que foi pesquisado diretamente o nome do pacote, para facilitar a captura dos dados.

Depois de obter acesso ao diretório, pode-se escolher os sub-diretórios a serem analisados. Os de maior relevância, neste caso, são: */app_parts* e */databases* [Tamma e Tindall 2015]. A partição */databases* contém dois bancos de dados com informações essenciais, o *mmssms.db* e *telephony.db*, que serão analisados a seguir.

O sub-diretório */app_parts* contém anexos que foram enviados e também recebidos como uma MMS. Com relação ao banco de dados *telephony.db*, este é relativamente pequeno, se comparado a outros, mas contém uma fonte de informação potencialmente útil. A tabela com informações mais significativas deste é a *siminfo*, que armazena um histórico de dados dos cartões SIM (Subscriber Identity Module – cartão utilizado para identificar, controlar e armazenar dados de telefones celulares) que foram utilizados no dispositivo.

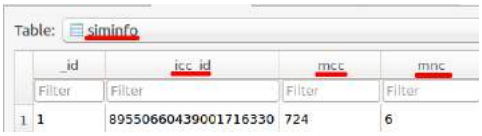


Table: siminfo

	_id	icc_id	mcc	mnc
	Filter	Filter	Filter	Filter
1	1	89550660439001716330	724	6

Figura 4.71: Informações da tabela *siminfo*

A Figura 4.71 mostra as informações contidas na tabela *siminfo* do dispositivo analisado. Nela, identifica-se o *Integrated Circuit Chip Card* - ICCID que é o número serial único dos aparelhos celulares. Também podem ser identificados o MMC - *Mobile Country Code*, que é um parâmetro para definir informações importantes sobre o país, operadora e região geográfica e o MNC - *Mobile Network Code* que é um código de dois ou três dígitos que é usado juntamente com o MMC para identificar uma operadora de telefonia [Brazileiro]. Os valores encontrados no dispositivo foram: para o ICCID, '89550660439001716330' (único mundialmente); para o MCC, '724' (onde 7 identifica o continente, América do Sul e América Central e 24 identifica o país, Brasil [Brazileiro]) e para o MNC, '6' que identifica que a operadora utilizada pelo dispositivo foi a VIVO [VIVO].

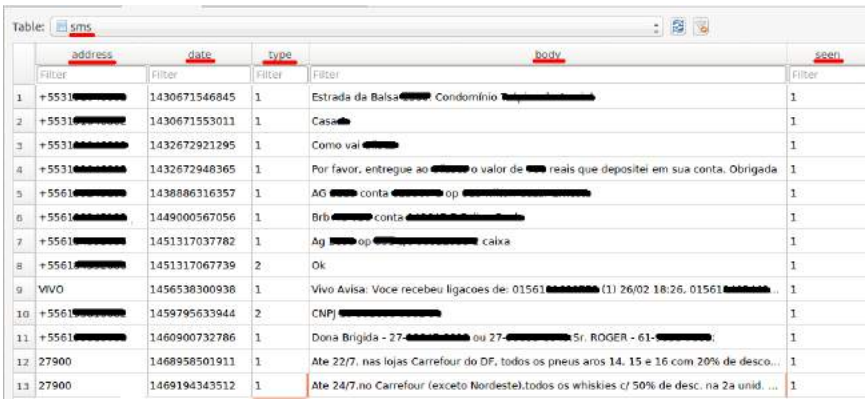


Table: sms

	address	date	type	body	seen
	Filter	Filter	Filter	Filter	Filter
1	+5531[redacted]	1430671546845	1	Estrada da Balsa[redacted] Condomínio [redacted]	1
2	+5531[redacted]	1430671553011	1	Casa[redacted]	1
3	+5531[redacted]	1432672921295	1	Como vai [redacted]	1
4	+5531[redacted]	1432672948365	1	Por favor, entregue ao [redacted] o valor de [redacted] reais que depositei em sua conta. Obrigada	1
5	+5561[redacted]	1438886316357	1	AG [redacted] conta [redacted] op [redacted]	1
6	+5561[redacted]	1449000567056	1	Brb [redacted] conta [redacted]	1
7	+5561[redacted]	1451317037782	1	Ag [redacted] op [redacted] caixa	1
8	+5561[redacted]	1451317067739	2	Ok	1
9	VIVO	1456538300938	1	Vivo Avisa: Voce recebeu ligacoes de: 01561[redacted] (1) 26/02 18:26, 01561[redacted]	1
10	+5561[redacted]	1459795633944	2	CNPJ [redacted]	1
11	+5561[redacted]	1460900732786	1	Dona Brigida - 27-[redacted] ou 27-[redacted] Sr. ROGER - 61-[redacted]	1
12	27900	1468958501911	1	Ate 22/7, nas lojas Carrefour do DF, todos os pneus aros 14, 15 e 16 com 20% de desco...	1
13	27900	1469194343512	1	Ate 24/7,no Carrefour (exceto Nordeste),todos os whiskies c/ 50% de desc. na 2a unid. ...	1

Figura 4.72: Informações da tabela *sms*

O banco de dados *mmssms.db* contém as informações relacionadas aos SMS's e MMS's e as principais tabelas de informações são as seguintes [Tamma e Tindall 2015]:

- ***part*:** contém informações sobre arquivos anexados a um MMS. Cada mensagem terá pelo menos duas partes, um cabeçalho e o anexo. Isto pode ser visto nas colunas *mid* e *ct*, assim como o tipo do arquivo anexado. A coluna *_data* provê o caminho para encontrar o arquivo no dispositivo;
- ***pdu*:** contém metadados sobre cada MMS e suas principais colunas são:
  - ***date*:** identifica quando a mensagem foi enviada ou recebida, no formato *Epoch* do Linux;
  - ***msg_box*:** mostra se a mensagem foi recebida, identificada com o número 1 ou se foi enviada, identificada com o número 2.
- ***sms*:** contém metadados sobre cada SMS e as colunas com informações essenciais são:
  - ***address*:** mostra o número de telefone do usuário remoto, independentemente se o SMS foi enviado ou recebido;
  - ***person*:** contém um valor que pode ser verificado no banco de dados *contacts2.db* na coluna *raw_contact_id* da tabela *data*. Estará com um espaço em branco se a mensagem tiver sido enviada do próprio dispositivo ou se tiver sido recebida por algum telefone não salvo na lista de contatos;
  - ***date*:** mostra o dia e hora que a mensagem foi enviada ou recebida no formato *Epoch*;
  - ***type*:** identifica se a mensagem foi recebida, tem '1' como valor ou se foi enviada, tendo como valor '2';
  - ***body*:** mostra o conteúdo das mensagens;
  - ***seen*:** indica se a mensagem foi lida ou não, se o valor for '0' a mensagem não foi lida e se for '1' a mensagem foi lida.

A Figura 4.72 mostra os resultados do dispositivo para cada uma das colunas descritas acima.

4.4.1.3 Análise do WhatsApp

Como foi visto no Capítulo 1, Seção 1.3, o WhatsApp é o *app* mais utilizado pelos brasileiros, portanto, é de extrema importância que seus dados sejam analisados pelo perito. O nome do pacote que armazena as informações referentes aos seus dados é o *com.whatsapp*. Ao pesquisar pelo pacote, vários sub-diretórios podem ser analisados. Os mais relevantes são: */files/*, */shared_prefs/* e claro, */databases/* [Tamma e Tindall 2015]. A Figura 4.73 mostra estes diretórios.

No diretório */files/*, há uma pasta chamada *Avatars/* que contém *thumbnails* das fotos de perfil de todos os contatos que utilizam o *app*. Além disso, possui um arquivo chamado *me* que contém o número de telefone associado com a conta e outro chamado *me.jpg*, que é uma versão de tamanho completo da foto de perfil do usuário.

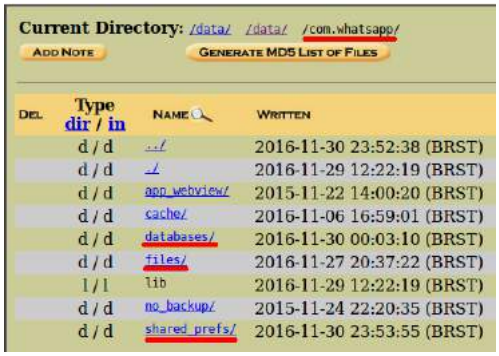


Figura 4.73: Diretórios do pacote do WhatsApp

O diretório */shared_prefs/* armazena dois arquivos importantes, o *RegisterPhone.xml* que armazena o número de telefone associado à conta do usuário e o *VerifySMS.xml* que mostra quando a conta foi verificada (no formato *Epoch* do Linux), indicando quando foi a primeira vez que o usuário começou a usar o aplicativo.

Quanto aos bancos de dados armazenados em */databases/*, dois contêm informações essenciais, o *msgstore.db* e o *wa.db*. O primeiro contém, como já diz no próprio nome, os dados de mensagens e suas principais tabelas são as seguintes [Tamma e Tindall 2015]:

- ***chat_list*:** mostra uma lista de todos os *chats* existentes no aplicativo. A coluna mais importante é a *key_remote_jid*, que exibe cada conta com a qual o usuário se comunicou. O valor na tabela é o número de telefone do usuário remoto. Por exemplo, se o número é *5561999999999@s.whatsapp.net*, o número do usuário remoto é *55-61-99999-9999*;
- ***group_participants*:** contém metadados sobre grupos de *chats*;
- ***media_refs*:** referência dos diretórios onde as mídias recebidas e enviadas estão armazenadas;
- ***messages*:** contêm os dados de mensagens. Suas principais colunas são:
  - ***key_remote_jid*:** exibe cada conta com a qual o usuário se comunicou;
  - ***key_from_me*:** indica o direcionamento da mensagem, se o valor for '0', ela foi recebida e se for '1', ela foi enviada;
  - ***data*:** contém os textos das mensagens;
  - ***timestamp*:** mostra o dia e hora que a mensagem foi recebida ou enviada (no formato *Epoch*).

  Se a mensagem tiver algum anexo:

  - ***media_mime_type*:** identifica o formato do arquivo;
  - ***media_size* e *media_name*:** mostram o tamanho e o nome das mídias;
  - ***media_caption*:** mostra a legenda do anexo, se este tiver uma;

– ***latitude*** e ***longitude*:** identificam a localização do anexo;

– ***thumb_ image*:** contém o caminho no qual o anexo está salvo no dispositivo;

– ***raw_ data*:** contém *thumbnails* para imagens e vídeos.

Quanto ao banco de dados *wa.db*, este é usado para armazenar informações sobre os contatos. Sua tabela mais importante é a *wa_ contacts*, que contém os nomes e números de telefone dos contatos, assim como o *status* de cada contato que for um usuário do WhatsApp (há uma coluna que determina se o contato é usuário do WhatsApp ou não, se tiver valor '1', é usuário).

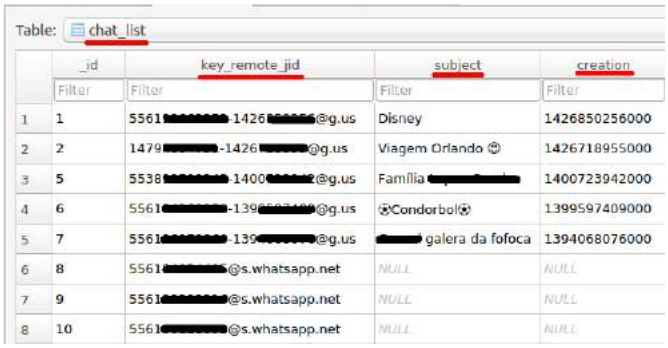


Table: chat_list

	_id	key_remote_jid	subject	creation
	Filter	Filter	Filter	Filter
1	1	5561[redacted]-1426[redacted]@g.us	Disney	1426850256000
2	2	1479[redacted]-1426[redacted]@g.us	Viagem Orlando 😍	1426718955000
3	5	5538[redacted]-1400[redacted]@g.us	Família [redacted]	1400723942000
4	6	5561[redacted]-139[redacted]@g.us	⚽Condorbol⚽	1399597409000
5	7	5561[redacted]-139[redacted]@g.us	[redacted] galera da fofoca	1394068076000
6	8	5561[redacted]@s.whatsapp.net	NULL	NULL
7	9	5561[redacted]@s.whatsapp.net	NULL	NULL
8	10	5561[redacted]@s.whatsapp.net	NULL	NULL

Figura 4.74: Informações da tabela *chat_ list*

A Figura 4.74 mostra a tabela *chat_ list*, observa-se que os números dos contatos são apresentados na segunda coluna. Em *subject* há duas opções de resultados a serem mostrados, se for um grupo, este terá seu nome apresentado e se for apenas um contato comum, estará como *null*. A mesma situação acontece em *creation*, onde apresenta-se a data de criação do grupo (formato *Epoch*).

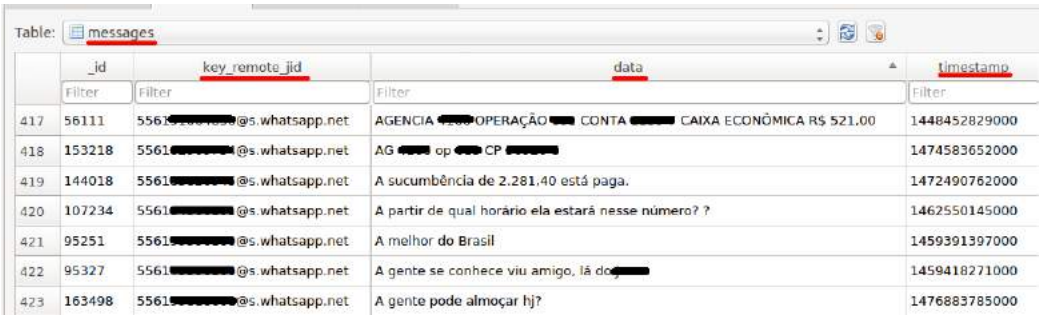


Table: messages

	_id	key_remote_jid	data	timestamp
	Filter	Filter	Filter	Filter
417	56111	5561[redacted]@s.whatsapp.net	AGENCIA [redacted] OPERAÇÃO [redacted] CONTA [redacted] CAIXA ECONÔMICA R$ 521.00	1448452829000
418	153218	5561[redacted]@s.whatsapp.net	AG [redacted] op [redacted] CP [redacted]	1474583652000
419	144018	5561[redacted]@s.whatsapp.net	A sucumbência de 2.281,40 está paga.	1472490762000
420	107234	5561[redacted]@s.whatsapp.net	A partir de qual horário ela estará nesse número? ?	1462550145000
421	95251	5561[redacted]@s.whatsapp.net	A melhor do Brasil	1459391397000
422	95327	5561[redacted]@s.whatsapp.net	A gente se conhece viu amigo, lá do[redacted]	1459418271000
423	163498	5561[redacted]@s.whatsapp.net	A gente pode almoçar hj?	1476883785000

Figura 4.75: Informações da tabela *messages*

A Figura 4.75 mostra a tabela *messages*, assim como na lista de *chats*, é possível visualizar claramente o número de telefone do contato. Na coluna *data*, se lê explicitamente o conteúdo das mensagens e em *timestamp* pode-se descobrir o dia e hora que a mensagem foi recebida ou enviada. A Figura 4.76 representa a mesma tabela, porém, nesta figura, apresenta-se o caso em que foram enviadas ou recebidas, mensagens de mídias. Pode-se, então, perceber as diferentes colunas que são preenchidas com informações sobre a mídia.

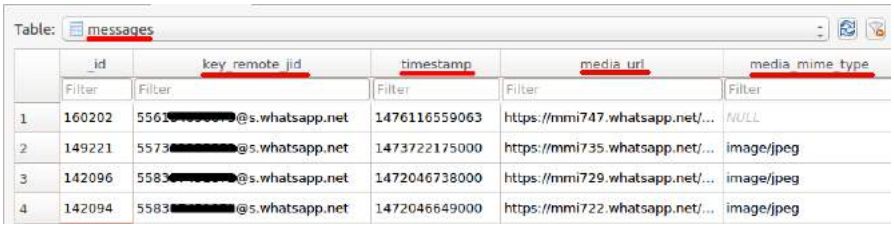


Figura 4.76: Informações da tabela *messages*

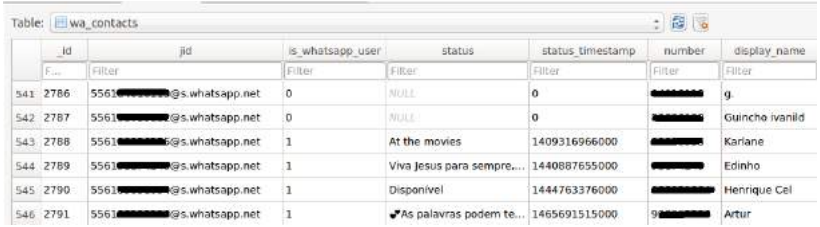


Figura 4.77: Informações da tabela *wa_contacts*

A Figura 4.77 representa a tabela *wa_contacts*. Neste caso, os contatos do telefone são apresentados e a coluna *is_whatsapp_user* identifica se o contato possui uma conta no WhatsApp ou não. Quando o usuário possui a conta, é apresentado o *status* de sua conta. Além disso, a coluna *display_name* mostra o nome do contato.

Para se analisar as mídias de áudio, vídeo e imagem, deve-se acessar o diretório */data/data/media/0/WhatsApp/Media/*. Neste, haverão pastas com os tipos de arquivos que podem ter sido armazenados no WhatsApp. A Figura 4.78 ilustra essa informação. Alguns arquivos (os que estiverem em vermelho) aparecem como deletados e, portanto, deve-se usar uma ferramenta de *carving* para recuperar estes arquivos.

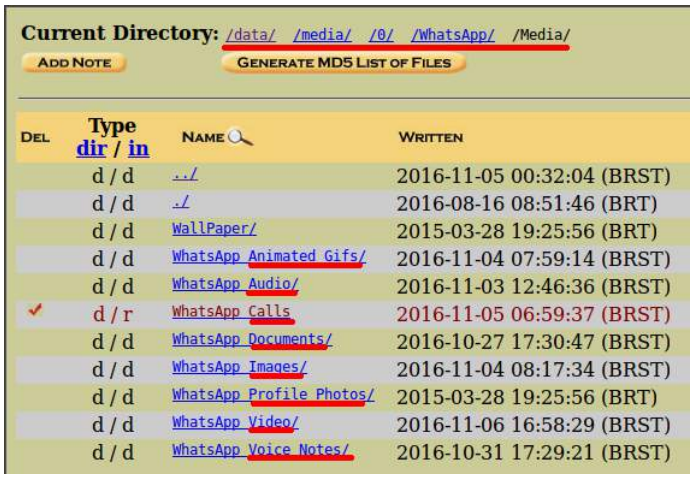


Figura 4.78: Diretório que armazena as mídias do WhatsApp

4.4.1.4 Análise do Facebook

O Facebook é um aplicativo de uma Rede Social muito utilizado pelos usuários de *smartphones*. O pacote responsável por armazenar os dados referentes a este é o *com.facebook.katana*. Seus principais diretórios são [Tamma e Tindall 2015]: */files/video-cache* que contêm vídeos do *feed* de notícias do usuário; o */cache/images* contém imagens do *feed* de notícias, assim como fotos de perfil dos contatos, e múltiplos sub-diretórios, que podem armazenar várias imagens relacionadas ao usuário e seus contatos e, por último, o */databases/* que será analisado a seguir.

Dentro do diretório */databases/*, são armazenados vários bancos de dados importantes para a análise do *app*. Neste trabalho serão discutidos os seguintes: *contacts_db2*, *newsfeed_db*, *notifications_db*, *pref_db* e *threads_db2*.

Com relação ao banco de dados *contacts_db2*, é sabido que ele contém informações sobre os contatos do usuário e possui uma tabela importante, a *contacts* que contém informações sobre os contatos e suas colunas mais importantes são:

- ***fbid*:** é um ID exclusivo que é usado para identificar o contato em outros bancos de dados;
- ***first_name***, ***last_name*** e ***display_name*:** mostram o nome do contato;
- ***small_picture_url***, ***big_picture_url*** e ***huge_picture_url***: contém *links* públicos para a imagem de perfil do contato;
- ***communication_rank*:** é um número identificando quantas vezes o contato se comunica com o usuário (levando em conta mensagens, comentários, entre outros); um número maior indica mais comunicação com esse contato;
- ***added_time_ms*:** mostra quando (no formato de época do Linux) o contato foi adicionado como amigo;
- ***bday_day*** e ***bday_month*:** mostram a data de nascimento do contato, mas não o ano;
- ***data*:** contém uma duplicata de todo o restante das informações no banco de dados, mas também contém o local do contato, que não é encontrado em lugar algum, à exceção deste mesmo.

O banco de dados *newsfeed_db* contém dados mostrados ao usuário em seu *newsfeed*. Dependendo do uso do aplicativo, ele pode ser um arquivo muito grande, contendo a tabela *home_stories*, que possui as seguintes colunas:

- ***fetched_at*:** mostra o horário em que a história foi extraída dos servidores do Facebook e, provavelmente, corresponde ao tempo que o usuário estava usando o aplicativo ou viu a história;
- ***story_data*:** contém a história armazenada como um *Binary Large Object* - BLOB de dados. Quando visualizado em um hexadecimal ou editor de texto, o nome de usuário da

pessoa que postar a história pode ser encontrado. O conteúdo da mensagem também pode ser encontrado em texto simples e é frequentemente precedido por uma *tag* que diz *text*.

O banco de dados *notifications_db* contém notificações enviadas ao usuário e armazenadas na tabela *gql_notifications*, cujas colunas são:

- ***seen_state*:** mostra se a notificação foi ou não vista e lida;
- ***updated*:** contém a hora em que a notificação foi atualizada no formato de *Epoch* do Linux;
- ***gql_payload*:** mostra o conteúdo da notificação, bem como o remetente, semelhante à coluna *story_data* no *newsfeed_db*. O conteúdo da mensagem é frequentemente precedido pelo parâmetro *text*;
- ***profile_picture_uri*:** contém uma URL pública para exibir a imagem do perfil do remetente;
- ***icon_url*:** possui um *link* para exibir o ícone associado à notificação.

O banco de dados *prefs_db* contém preferências de aplicativo armazenadas na tabela *preferences* e possui algumas linhas importantes:

- ***/auth/user_data/fb_username*:** mostra o nome de usuário do Facebook;
- ***/config/gk/last_fetch_time_ms*:** é o *timestamp* da última comunicação do aplicativo com os servidores do Facebook, mas pode não ser a hora exata da última interação do usuário com o aplicativo;
- ***/fb_android/last_login_time*:** mostra a última vez em que o usuário efetuou *login*, através do aplicativo. O banco de dados contém muitos outros *timestamps*. Quando reunidos, esses podem ser usados para construir um perfil do uso do aplicativo;
- ***/auth/user_data/fb_me_user*:** contém dados sobre o usuário, incluindo seu nome, endereço de e-mail e número de telefone.

O banco de dados *threads_db* contém informações de mensagens descritas na tabela *messages* e possui as colunas:

- ***msg_id*:** cada mensagem tem um ID exclusivo;
- ***text*:** contém a mensagem em texto simples;
- ***sender*:** identifica o Facebook ID e o nome do remetente da mensagem;
- ***timestamp_ms*:** é a hora em que a mensagem foi enviada, no formato *Epoch* do Linux;
- ***attachments*:** contém uma URL pública para recuperar as imagens anexadas;

- ***coordinates*:** terá a latitude e a longitude do remetente se eles optarem por mostrar sua localização;
- ***source*:** identifica se a mensagem foi enviada através do *site* ou do aplicativo.

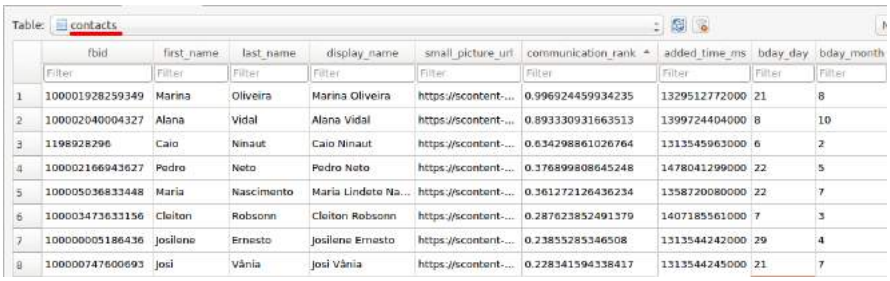


Table: contacts

	fbid	first_name	last_name	display_name	small_picture_url	communication_rank	added_time_ms	bday_day	bday_month
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	100001928259349	Marina	Oliveira	Marina Oliveira	https://scontent-...	0.996924459934235	1329512772000	21	8
2	100002040004327	Alana	Vidal	Alana Vidal	https://scontent-...	0.893330931663513	1399724404000	8	10
3	1198928296	Caio	Ninaut	Caio Ninaut	https://scontent-...	0.634298861026764	1313545963000	6	2
4	100002166943627	Pedro	Neto	Pedro Neto	https://scontent-...	0.376899808645248	1478041299000	22	5
5	100005036833448	Maria	Nascimento	Maria Lindete Na...	https://scontent-...	0.361272126436234	1358720080000	22	7
6	100003473633156	Cleiton	Robsonn	Cleiton Robsonn	https://scontent-...	0.287623852491379	1407185561000	7	3
7	100000005186436	Josilene	Ernesto	Josilene Ernesto	https://scontent-...	0.23855285346508	1313544242000	29	4
8	100000747600693	Josi	Vânia	Josi Vânia	https://scontent-...	0.228341594338417	1313544245000	21	7

Figura 4.79: Informações da Tabela *contacts*

A Figura 4.79 demonstra a tabela *contacts* do dispositivo.

4.4.1.5 Análise do Snapchat

O Snapchat é um aplicativo que oferece um serviço de compartilhamento de imagens e vídeos e também troca de mensagens. Seu principal diferencial é o fato de que as imagens e vídeos são deletadas após um tempo limite configurado pelo usuário, de 1 a 10 segundos. Se um usuário tirar um *screenshot* da imagem, o usuário que enviou a mesma é notificado. As mensagens de texto não têm um tempo de expiração [Tamma e Tindall 2015].

O pacote que armazena as informações do *app* é o *com.snapchat.android*. Nele, há alguns arquivos importantes que serão descritos a seguir:

- ***/cache/stories/received/thumbnail*:** contém *thumbnails* de imagens tiradas pelo usuário, mas os arquivos podem não ter extensões de arquivo apropriadas;
- ***/shared_prefs/com.snapchat.android_preferences.xml*:** contém o endereço de e-mail usado para criar uma conta e o número de telefone do dispositivo registrado na conta;
- ***/databases/tcspahn.db*:** banco de dados que contém as outras informações sobre o uso do *app*. Possui algumas tabelas importantes:
  - ***chat*:** lista os chats mostrando quem enviou, recebeu e quando a mensagem foi enviada ou recebida;
  - ***ContactsOnSnapchatTable*:** mostra os usuários da agenda de contatos que também tem o *app* instalado. A coluna *isAddedAsFriends* mostra o valor '1' se o usuário tiver sido adicionado como contato;
  - ***conversation*:** tem informações sobre cada *chat* aberto. Inclui o usuário que enviou, recebou e quando foi a última vez que *snaps* foram enviados ou recebidos por cada usuário;

– ***Friends*:** inclui todos os usuários que foram adicionados como "amigo" e também mostra quando o contato foi adicionado;

– ***ReceivedSnaps*:** contém dados sobre imagens e vídeos recebidos, informa quando foi recebido, o *status* do *snap*, se um *screenshot* foi tirado e também quem enviou;

– ***SentSnaps*:** contém dados sobre imagens e vídeos enviados, informa quando foi enviado, o *status* do *snap*, se um *screenshot* foi tirado e também quem recebeu.

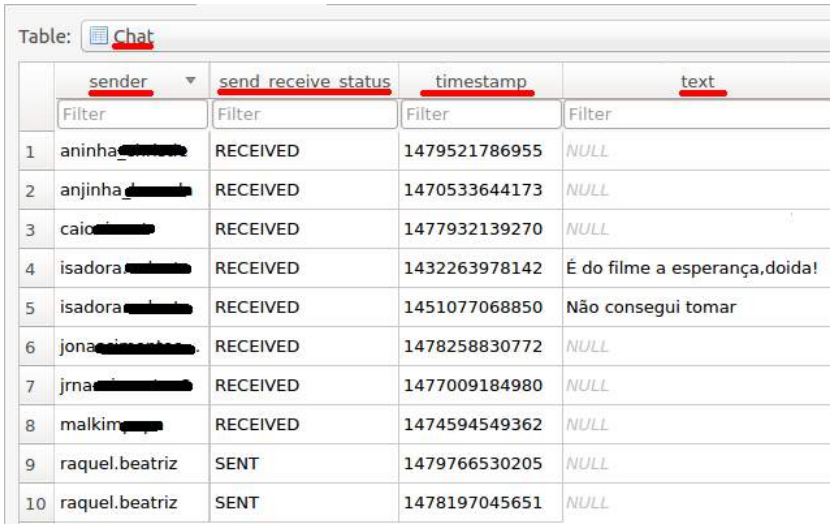


Table: Chat

	sender	send_receive_status	timestamp	text
	Filter	Filter	Filter	Filter
1	aninha[redacted]	RECEIVED	1479521786955	NULL
2	anjinha_[redacted]	RECEIVED	1470533644173	NULL
3	caio[redacted]	RECEIVED	1477932139270	NULL
4	isadora[redacted]	RECEIVED	1432263978142	É do filme a esperança,doida!
5	isadora[redacted]	RECEIVED	1451077068850	Não consegui tomar
6	jona[redacted]	RECEIVED	1478258830772	NULL
7	jrna[redacted]	RECEIVED	1477009184980	NULL
8	malkim[redacted]	RECEIVED	1474594549362	NULL
9	raquel.beatriz	SENT	1479766530205	NULL
10	raquel.beatriz	SENT	1478197045651	NULL

Figura 4.80: Informações da tabela *Chat*

A Figura 4.80 mostra a tabela *Chat* do dispositivo, informa quem enviou a mensagem, o *status* (se foi enviada ou recebida), quando esta transferência aconteceu e o conteúdo da mensagem.

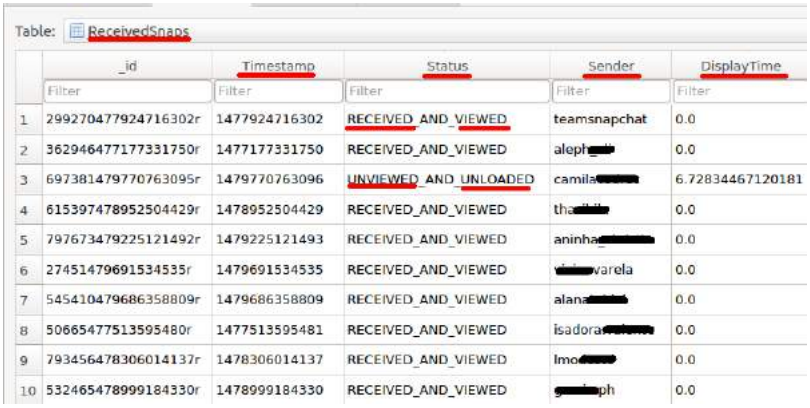


Table: ReceivedSnaps

	_id	Timestamp	Status	Sender	DisplayTime
	Filter	Filter	Filter	Filter	Filter
1	299270477924716302r	1477924716302	RECEIVED_AND_VIEWED	teamsnapchat	0.0
2	362946477177331750r	1477177331750	RECEIVED_AND_VIEWED	aleph[redacted]	0.0
3	697381479770763095r	1479770763096	UNVIEWED_AND_UNLOADED	camila[redacted]	6.72834467120181
4	615397478952504429r	1478952504429	RECEIVED_AND_VIEWED	tha[redacted]	0.0
5	797673479225121492r	1479225121493	RECEIVED_AND_VIEWED	aninha[redacted]	0.0
6	27451479691534535r	1479691534535	RECEIVED_AND_VIEWED	[redacted]varela	0.0
7	5454104796863588O9r	1479686358809	RECEIVED_AND_VIEWED	alana[redacted]	0.0
8	50665477513595480r	1477513595481	RECEIVED_AND_VIEWED	isadora[redacted]	0.0
9	793456478306014137r	1478306014137	RECEIVED_AND_VIEWED	lmo[redacted]	0.0
10	532465478999184330r	1478999184330	RECEIVED_AND_VIEWED	[redacted]ph	0.0

Figura 4.81: Informações da tabela *ReceivedSnaps*

Já a Figura 4.81 ilustra a tabela *ReceivedSnaps*, exibe quando o *snap* foi recebido, o *status*, se foi recebido e visualizado ou se não foi visto e nem carregado, o usuário que enviou e a duração do *snap*.

4.4.2 Caso 2 - Dispositivo ligado, com bloqueio de tela, sem acesso ADB e sem permissões de *root*

Se o dispositivo estiver com um bloqueio de tela, o primeiro passo a se tomar é conectar o aparelho ao computador, via cabo USB. Este procedimento deve ser tomado, pois o acesso ADB pode estar ativado e algumas análises já poderiam ser feitas, se este fosse o caso. Ao se deparar sem o acesso ADB, o investigador pode utilizar força bruta para quebrar o padrão de bloqueio.

O método que foi utilizado foi o de substituição da partição *recovery* original do dispositivo [Lellis e Cruz 2015] e também utilizou-se um programa para efetuar o desbloqueio [XDADevelopers]. Existem várias imagens da partição *recovey* modificadas que podem ser utilizadas. A imagem utilizada neste caso, foi a do *Team Win Recovery Project* - TWRP por ter uma interface sensível ao toque e personalizável [TWRP]. É importante que se escolha uma versão compatível ao dispositivo que estiver sendo utilizado.

O TWRP é um projeto comunitário *open source*. Uma partição *recovery* personalizada é utilizada para instalar *softwares* próprios no dispositivo. Estes *softwares* podem incluir modificações menores, como fazer o *root* do aparelho ou até mesmo substituir o seu *firmware* com uma ROM completamente individualizada [TWRP].

Depois de feito o *download*, estará compactado e deve-se, então, descompactá-lo. Ao descompactar, será encontrado o arquivo *twrp-2.8.1.0-I9515.tar.md5* (o nome dependerá da versão que foi obtida). Este arquivo deve ser também descompactado (antes, exclui-se o termo *.md5*) para que seja obtido o *recovery.img* que substituirá posteriormente o arquivo original do dispositivo. Para transferir será utilizada novamente a ferramenta *Heimdall*.

O próximo passo é exatamente o mesmo que foi observado na Seção 4.4.1, o dispositivo tem que estar em modo *debug* para que o arquivo *recovery.img* seja instalado no mesmo. Para conferir se a instalação foi feita corretamente, pode-se colocar o dispositivo em modo *recovery* (apertando as teclas volume para cima + botão central + botão de ligar, ao mesmo tempo). Após este procedimento, foi feito o *download* de uma ferramenta que burla qualquer tipo de tela de bloqueio de dispositivos Samsung [XDADevelopers]. Para instalar esta no dispositivo, foi utilizado um cartão de memória com a mesma copiada. Usou-se a interface do TWRP para completar a instalação.

As Figuras 4.82 e 4.83 ilustram os próximos procedimentos que devem ser feitos. Ao acessar o modo *recovery* com o TWRP, pode-se ir até a opção *Install*. Ao entrar nesta, procura-se pelo cartão SD e pelo programa que foi colocado no mesmo. Ao clicar no *software*, deve-se deslizar a tela indicada para que a operação seja finalizada. Quando ela finalizar com sucesso, deve-se fazer um *reboot* no sistema. **Assim que o dispositivo religar, ele já estará sem a tela de bloqueio!**

Após estes passos, pode-se ativar a Depuração USB, mas ainda precisa-se da função de *root*, sendo assim, a partir daqui, os procedimentos serão os mesmos da Seção 4.4.1.

Figura 4.82: Instruções do procedimento de instalação da nova ROM

Figura 4.83: Instruções do procedimento de instalação da nova ROM

4.4.3 Caso 3 - Dispositivo ligado, com bloqueio de tela, com acesso ADB e com permissões de *root*

Ao deparar com esta situação, o perito pode, primeiramente, testar o acesso de Depuração USB. Como este está ativado, ele pode tentar fazer as operações de imagem de partições. Se o dispositivo não tivesse com permissões de *root*, os procedimentos da Seção 4.4.1 já poderiam ser realizados. Porém, ao constatar que as opções de *root* também estão ativadas, ele já pode fazer todos os exames necessários sem nenhuma outra intervenção.

Se for preciso acessar o dispositivo para uma extração manual, o procedimento da Seção 4.4.2 pode ser realizado. E, neste caso, não seria necessário o uso de um cartão de memória, a função *push* do ADB pode ser utilizada para transferir o programa que foi utilizado para burlar o bloqueio de tela diretamente para o dispositivo móvel.

4.5 Análise de Arquivos Deletados com o *Scalpel* e *Foremost*

Para recuperar os arquivos deletados que são apresentados em vermelho no *Autopsy*, pode-se utilizar duas ferramentas: *Scalpel* e *Foremost*. Para o teste, utilizou-se a mesma partição */data* e os mesmos parâmetros de pesquisa. Os formatos de arquivos que foram configurados para serem recuperados são: JPG, PNG, BMP, MPG, PDF e DOC. A seguir, serão apresentados os procedimentos tomados e os resultados das análises. As Figuras 4.84 e 4.85 mostram o procedimento para a captura dos dados deletados.

Utilizou-se um HD externo para salvar as informações coletadas com as ferramentas. O perito também pode utilizar este método para armazenar provas separadamente e para facilitar a elaboração do relatório final.

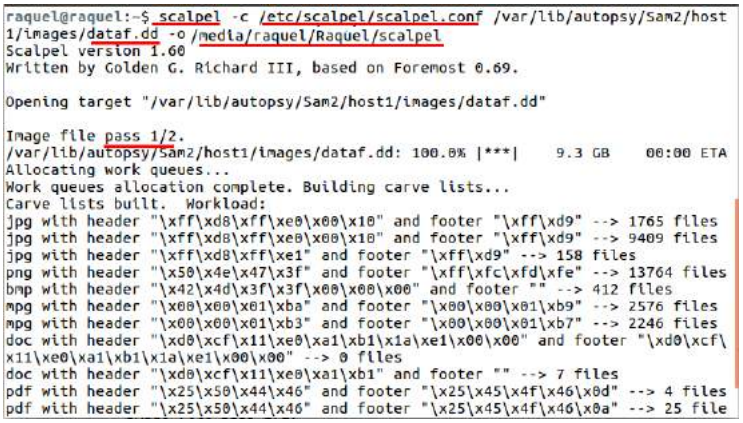


Figura 4.84: Procedimento para recuperação de dados deletados com o *Scalpel*

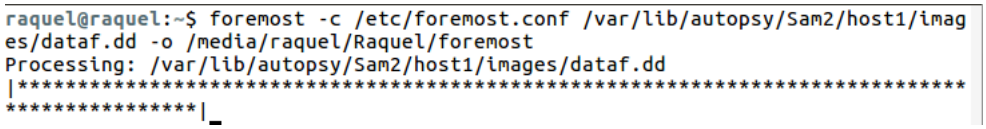


Figura 4.85: Procedimento para recuperação de dados deletados com o *Foremost*

Nas Figuras 4.86 e 4.87 são apresentadas a quantidade de arquivos recuperados pelas ferramentas. Vale lembrar que não há garantia de que os arquivos estarão totalmente executáveis, porém, neste caso, a maior parte das imagens JPG estavam intactas nos resultados de ambos *softwares*.

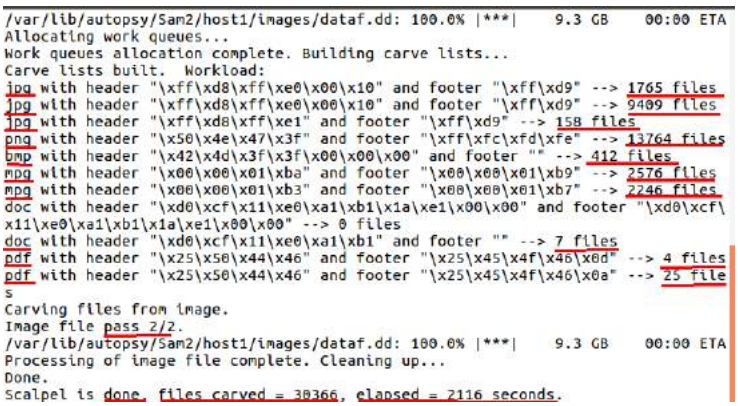


Figura 4.86: Resultado do *Scalpel*

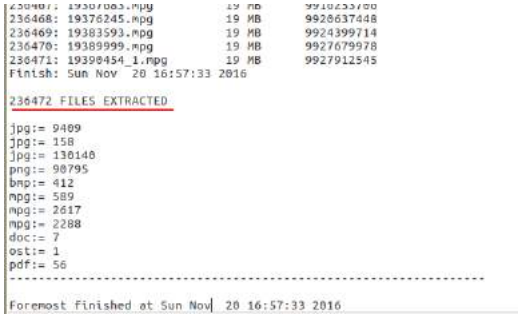


Figura 4.87: Resultado do *Foremost*

Tabela 4.2: Comparação de resultados entre o *Scalpel* e o *Foremost*

Arquivos Recuperados	*Scalpel*	*Foremost*
JPG	11.332	139.707
PNG	13.764	90.795
BMP	412	412
MPG	4.822	5.494
DOC	7	7
PDF	29	56
Total	30.336	236.472

A Tabela 4.2 faz uma comparação entre os resultados obtidos e, neste caso, a ferramenta *Foremost* conseguiu recuperar uma quantidade muito superior de dados com relação ao *Scalpel*. Mas, apesar de ter encontrado mais PDF's, por exemplo, obteve-se o mesmo número de arquivos PDF não corrompidos que o *Scalpel* e apenas três deles foram recuperados de uma totalidade de 100%.

4.6 Comparações de Resultados

Após as análises, são apresentadas nas Tabelas 4.3 e 4.4, comparações entre os cenários utilizados e entre as ferramentas. No Capítulo 5 serão discutidos em quê estes resultados implicaram.

Tabela 4.3: Comparação entre os diferentes cenários

Etapas dos processos	Android AVD	Android *Smartphone*
Conhecimento detalhado sobre a exata versão do SO e sua configuração	SIM	SIM
Módulos têm que ser compilados contra o *kernel* do dispositivo alvo	SIM	SIM
Cross compilation sempre necessária	SIM	SIM
Pesquisa sobre as fontes e configurações do *kernel*	NÃO	SIM
kernel tem de ser compilado para se ter os símbolos do mesmo	NÃO	SIM
Achar versão do compilador compatível	SIM	SIM
Tela de bloqueio tem que ser superada	NÃO	SIM
Software tem de ser compilado para cada alvo	SIM	SIM
Requer permissões de *root*	SIM	SIM
Rooting	NÃO	SIM

Tabela 4.4: Comparação entre as diferentes ferramentas

Etapas do Processo	LiME	*Volatility*	TSK/ *Autopsy*	*Scalpel*/ *Foremost*	*AFLogical OSE*
Conhecimentos sobre o *kernel* do dispositivo	SIM	SIM	NÃO	NÃO	NÃO
Software tem de ser compilado	SIM	SIM	NÃO	NÃO	SIM, tem de ser instalado no dispositivo
Requer permissões de *root*	SIM	SIM	SIM	SIM	SIM
Tela de bloqueio tem que ser superada	Se a Depuração USB estiver ativa, NÃO Se a Depuração USB estiver inativa, SIM				SIM
Análise específica para cada modelo	SIM	SIM	NÃO	NÃO	NÃO
Utilização de outros mecanismos para fazer a imagem que será analisada	NÃO	SIM, o LiME	SIM, dd	SIM, dd	NÃO
Requer acesso ADB	SIM	NÃO	SIM	NÃO	SIM

4.7 Criação do *Live CD*

A partir do momento em que se obteve um conjunto de ferramentas já configuradas e pré-estabelecidas para se fazer uma análise forense, pode-se realizar a criação de um *Live CD*. O *Live CD* é um CD, ou dispositivo externo USB, que contém um sistema operacional que não precisa ser instalado no computador do usuário para que o mesmo seja utilizado.

A ferramenta *Systemback* foi utilizada para realizar o procedimento de criação do sistema *live*. Esta é considerada um dos melhores mecanismos para se fazer *backup* em sistemas operacionais Linux, por sua facilidade de uso e outros recursos. Dentre estes recursos, destacam-se a opção de copiar completamente o sistema, podendo incluir os arquivos de usuário ou não e, depois, realizar a restauração para uma determinada data anterior [Simioni]. Outra função é a de criar uma imagem inicializável, que foi utilizada nesta pesquisa. As instruções de instalação da ferramenta encontram-se no Anexo I.

A Figura 4.88 mostra o menu principal da ferramenta e em vermelho se encontram a partição */home*, que é o diretório em que a imagem será salva e também a opção *Live system create*, que foi utilizada para criar a imagem do sistema.

A Figura 4.89 exibe como pode-se realizar o procedimento, o nome do sistema pode ser escolhido na opção *Name of the Live system* e então, depois de escolhido, cria-se a imagem na opção *Create new*. Para que as ferramentas e configurações específicas utilizadas durante o trabalho fossem mantidas, assim como os roteiros e livros, foi marcada a opção *Include the user data fi-*

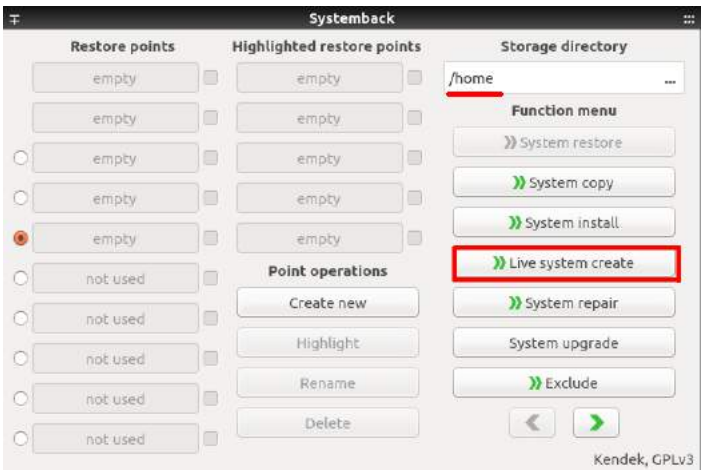


Figura 4.88: Menu principal da ferramenta *systemback*

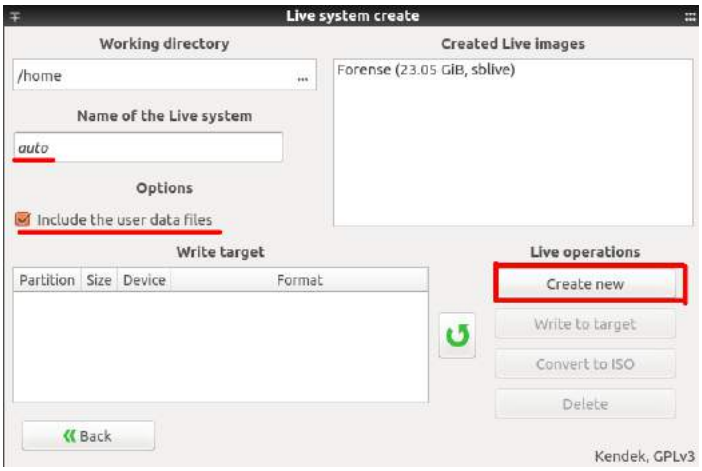


Figura 4.89: Opção para a criação do sistema *live*

Figura 4.90: Processo de criação

les. O progresso da operação é mostrado na Figura 4.90 e pode demorar um tempo considerável, principalmente se a opção de manter os dados de usuário for selecionada.

A Figura 4.91 exibe como a imagem criada pode ser transferida para um determinado alvo, sendo este um CD ou dispositivo USB. Após a criação da imagem, a mesma estará listada (neste caso, a imagem foi a *Forense (23.05 GiB, sblive)*). Assim, ao clicar na mesma e com o dispositivo

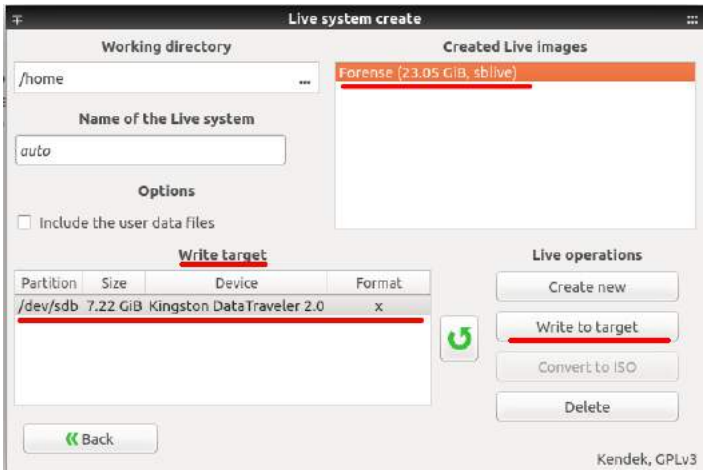


Figura 4.91: Opção para escrever a imagem criada para o dispositivo USB

externo inserido no computador, a opção *Write to target* aparecerá. Esta opção fará o processo de escrita automaticamente e, ao final, o *Live CD* estará criado e pronto para ser utilizado em qualquer computador.

Capítulo 5

Conclusão

Nesta pesquisa, foram analisadas as ferramentas que realizam uma análise forense em dispositivos móveis para a criação de um *Live CD*. Foram utilizados vários cenários, variando-se o método de extração, análises e a versão do sistema operacional Android utilizado. Com os resultados obtidos para cada caso, foram criadas tabelas com a comparação dos mesmos. Neste capítulo são apresentadas as conclusões obtidas, a partir da análise realizada.

Das tabelas apresentadas no Capítulo 4, infere-se que para estudos laboratoriais, o emulador do Android Studio é eficiente, pois alguns dos processos que mais demandam técnica do perito, como o de se encontrar o *kernel* do dispositivo e o *rooting*, não são necessários para um AVD. Enquanto em um dispositivo real, os procedimentos exigem um maior conhecimento e algumas técnicas de força bruta por parte do analista.

As ferramentas analisadas apresentam seus resultados de diferentes maneiras. Enquanto o *Volatility* analisa a imagem da memória volátil, capturada com o LiME, com o auxílio de linhas de comando, o *Autopsy* apresenta uma interface gráfica que facilita a visualização dos resultados da memória persistente do dispositivo. O *Scalpel* e o *Foremost* são algumas das alternativas para se recuperar dados deletados de um dispositivo, ambos possuem uma margem satisfatória de informações restauradas, e foi possível perceber que, arquivos de imagens são aqueles que têm maior possibilidade de recuperação, com o uso de *file carving*. E a ferramenta *AFLogical OSE*, pode ser uma alternativa para se obter apenas resultados pontuais, que é o caso dos contatos, chamadas, SMS's e MMS's.

Com relação aos aplicativos examinados com o auxílio do *Autopsy*, percebeu-se que são armazenadas informações sensíveis dos usuários de *smartphones* e também de seus contatos. Dados pessoais foram facilmente encontrados com a análise da partição */data* dos dispositivos e os *apps* que deveriam proteger aquilo que é compartilhado e guardado, permitiram que, em uma simples análise em seus bancos de dados, tudo aquilo que foi realizado no mesmo, pudesse ser explorado.

Quanto à cadeia de custódia, os procedimentos de compilação do *kernel*, *rooting* ou remoção da tela de bloqueio, não fazem alterações na partição */data*, que é onde estão armazenados os dados do usuário. Estes procedimentos se restringem ao processo de carga do S.O., modificando, por exemplo, as partições */boot* e */recovery*, isto é, a integridade do dispositivo, com relação às

informações que têm de ser coletadas, não é afetada.

Como consequência, o *Live CD* foi criado e ficou pronto para ser utilizado. As ferramentas utilizadas neste trabalho foram instaladas no mesmo, da mesma forma que alguns livros e roteiros que possibilitaram uma melhor compreensão por parte do usuário também o foram. Além de ser utilizado como um sistema *live*, foi possível fazer a instalação do CD em um computador e, a partir dele, executar as análises forenses.

Com os resultados obtidos, foi possível perceber que um dos maiores desafios da perícia em dispositivos móveis, com ferramentas *open source*, é o fato de que os dispositivos precisam estar com a função ADB ativada e do *root* habilitado. Cada dispositivo a ser analisado, tem uma diferente forma de obtenção das funções de super usuário e, por isso, dificulta o trabalho da perícia. Além disso, os diversos padrões de tela de bloqueio, podem complicar o andamento de uma investigação, já que, para ativação da função de depuração USB (acesso ADB) o perito deve ter acesso manual ao telefone celular.

Como pesquisas futuras sugeridas após o desenvolvimento deste trabalho, pode-se incluir a criação de ferramentas *open source* que façam o processo de *rooting* nos dispositivos e, ao mesmo tempo, possibilitem, no mesmo ambiente, a realização da análise das partições escolhidas pelo perito. Isto auxiliaria em resultados que poderiam ser obtidos mais rapidamente, com mais eficácia e preservação do dispositivo.

Assim, foi possível encontrar ferramentas que, com baixo custo, efetuaram a análise do dispositivo da mesma maneira que uma ferramenta com foco comercial realizaria. Algumas técnicas a mais foram utilizadas, e, com elas, chegou-se ao resultado final esperado: a coleta dos dados do usuário armazenados no smartphone, conservando a integridade do mesmo.

REFERÊNCIAS BIBLIOGRÁFICAS

[Autopsy]AUTOPSY. *Autopsy: Description*. Acessado em Outubro de 2016. Disponível em: <http://www.sleuthkit.org/autopsy/v2/>.

[Brazileiro]BRAZILEIRO, I. *Conceitos de MCC e MNC*. Acessado em Novembro de 2016. Disponível em: <http://www.imm-tecnologia.com.br/o-que-e-mcc-e-mnc-da-tim-claro-vivo-e-oi/>.

[Buchanan]BUCHANAN, B. *Scalpel Configuration File*. Acessado em Novembro de 2016. Disponível em: <https://asecuritysite.com/scalpel.conf.txt>.

[Case]CASE, A. *Recoving tmpfs from Memory with Volatility*. Acessado em Outubro de 2016. Disponível em: <http://memoryforensics.blogspot.com.br/2012/08/recoving-tmpfs-from-memory-with.html>.

[CF-Auto-Root]CF-AUTO-ROOT. *CF-Auto-Root Download*. Acessado em Novembro de 2016. Disponível em: <https://download.chainfire.eu/547/CF-Root/CF-Auto-Root/CF-Auto-Root-jfvelte-jfvelteub-gti9515l.zip>.

[Chirgwin]CHIRGWIN, R. *Permission Android*. Acessado em Novembro de 2016. Disponível em: <http://www.theregister.co.uk/2014/01/07/app_to_manage_android_app_permissions/>.

[Chung]CHUNG, J. *Compile and Run the Android Goldfish Kernel*. Acessado em Outubro de 2016. Disponível em: <http://rockhopper.monmouth.edu/cs/jchung/cs438/cs-438_compile_and_run_the_android_goldfish_kernel>.

[Eason]EASON, J. *Android M Developer Preview Tools*. Acessado em Novembro de 2016. Disponível em: <http://android-developers.blogspot.com.br/2015/05/android-m-developer-preview-tools.html>.

[eMarketer]EMARKETER. *Slowing Growth Ahead for Worldwide Internet Audience*. Acesso em Outubro de 2016. Disponível em: <http://www.emarketer.com/Article/Slowing-Growth-Ahead-Worldwide-Internet-Audience/1014045>.

[Epoch]EPOCH. *Epoch Converter - Unix Timestamp Converter*. Acessado em Novembro de 2016. Disponível em: <http://www.epochconverter.com/>.

[Freetz]FREETZ. *Build Error*. Acessado em Outubro de 2016. Disponível em: <https://freetz.org/ticket/2759>.

[Gartner]GARTNER. *Gartner Says Five of Top 10 Worldwide Mobile Phone Vendors Increased Sales in Second Quarter for 2016*. Acesso em Outubro de 2016. Disponível em: <http://www.gartner.com/newsroom/id/3415117>.

[Hex]HEX, B. *BLess Hex Editor*. Acessado em Outubro de 2016. Disponível em: <http://home.gna.org/bless/>.

[IBOPE]IBOPE. *IBOPE-Aplicativos mais utilizados pelos brasileiros*. Acesso em Novembro de 2016. Disponível em: <http://www.ibope.com.br/pt-br/noticias/Paginas/WhatsApp-e-o-aplicativo-mais-usado-pelos-internautas-brasileiros.aspx>.

[Lellis e Cruz 2015]*Extração de dados em smartphones com sistema Android usando substituição da partição recovery*. 36–45 p.

[Ligh et al. 2014]LIGH, M. H. et al. The art of memory forensics. In: *The Volatility Framework*. [S.l.]: Wiley, 2014. p. 55–56.

[LiME]LIME. *GitHub - 504ensicsLabs/LiME*. Acessado em Outubro de 2016. Disponível em: <https://github.com/504ensicsLabs/LiME>.

[Lohrum]LOHRUM, M. *Live Imaging an Android device*. Acessado em Novembro de 2016. Disponível em: <http://freeandroidforensics.blogspot.com.br/2014/08/live-imaging-android-device.html>.

[MobileResearch]MOBILERESEARCH. *Netcat binaries compiled for ARM that work on Android devices*. Acessado em Outubro de 2016. Disponível em: <https://github.com/MobileForensicsResearch/netcat>.

[Murphy]MURPHY, D. C. Cellular phone evidence - data extraction & documentation. Unpublished Article.

[Neukamp]NEUKAMP, P. A. *Formulario de Cadeia de Custódia*. Acessado em Novembro de 2016. Disponível em: <http://fdtk.com.br/wiki/tiki-index.php?page=formulario>.

[NowSecure]NOWSECURE. *AFLogical-OSE*. Acessado em Outubro de 2016. Disponível em: <https://github.com/nowsecure/android-forensics>.

[Parkour]PARKOUR, M. *Contagio Mobile - Mobile Malware Mini Dump*. Acessado em Outubro de 2016. Disponível em: <http://contagiominidump.blogspot.com.br/>.

[Permissions]PERMISSIONS, G. I. S. *System Permissions | Android Developers*. Acessado em Novembro de 2016. Disponível em: <https://developer.android.com/guide/topics/security/permissions.html>.

[Santos]SANTOS, R. F. d. Ferramentas de computação forense baseadas em software livre. Unpublished Article.

[SDK]SDK, G. I. *Download Android Studio and SDK Tools*. Acessado em Março de 2016. Disponível em: <https://developer.android.com/studio/index.html>.

[Simioni]SIMIONI, D. *Systemback*. Acesso em Novembro de 2016. Disponível em: <http://www.diolinux.com.br/2013/06/como-instalar-o-systemback-no-ubuntu.html>.

[Simão 2011]SIMãO, A. M. de L. *Proposta de Método para Análise Pericial em Smartphone com Sistema Operacional Android*. Dissertação (Dissertação de Mestrado) — Universidade de Brasília, Setembro 2011.

[SleuthKit]SLEUTHKIT. *The Sleuth Kit: File and Volume System Analysis*. Acessado em Outubro de 2016. Disponível em: <http://www.sleuthkit.org/sleuthkit/desc.php>.

[SQLite]SQLITE. *DB Browser for SQLite*. Acessado em Novembro de 2016. Disponível em: <http://sqlitebrowser.org/>.

[Stephen]STEPHEN. *Busybox Download*. Acessado em Outubro de 2016. Disponível em: <http://busybox.br.uptodown.com/android>.

[Stirparo, Fovino e Kounelis 2013]STIRPARO, P.; FOVINO, I. N.; KOUNELIS, I. Data-in-use leakages from android memory - test and analysis. *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, IEEE, p. 718–725, 2013.

[Sylve]SYLVE, J. *Datalore: Android Memory Analysis*. Acessado em Outubro de 2016. Disponível em: <http://downloads.volatilityfoundation.org/omfw/2012/OMFW2012_Sylve.pdf>.

[Sylve et al. 2012]SYLVE, J. et al. Acquisition and analysis of volatile memory from android devices. *Digital Investigation*, Elsevier, v. 9, n. 3-4, p. 175–184, 2012.

[Tamma e Tindall 2015]TAMMA, R.; TINDALL, D. *Learning Android Forensics*. 1. ed. Birmingham, UK: Packt Publishing, 2015.

[TWRP]TWRP. *Team Win Recovery Project*. Acess0 em Novembro de 2016. Disponível em: <http://teamw.in/project/twrp2/>.

[Velho 2016]VELHO, J. A. o. Tratado de computação forense. In: *Exames em equipamentos portáteis e telefonia móvel*. [S.l.]: Millennium Editora, 2016. p. 313–340.

[Versions]VERSIONS, G. I. P. *Android Dashboards-Platform Versions*. Acesso em Novembro de 2016. Disponível em: <https://developer.android.com/about/dashboards/index.html>.

[VirusTotal]VIRUSTOTAL. *Virus Total*. Acessado em Outubro de 2016. Disponível em: <https://virustotal.com/>.

[VIVO]VIVO. *GSM Association Roaming Database, Structure and Updating Procedures - Vivo*. Acessado em Novembro de 2016. Disponível em: <https://www.vivo.com.br/portalweb/ShowPropertyServlet?nodeId=/UCMRepository/CONTRIB_093689>.

[Volatility]VOLATILITY. *Volatility Wiki*. Acessado em Outubro de 2016. Disponível em: <https://github.com/volatilityfoundation/volatility/wiki>.

[Wei et al. 2012]*Permission Evolution in the Android Ecosystem.* [S.l.]: ACM, 2012. 31–40 p.

[Wilson]WILSON, C. *Android Phone Forensic Analysis – Unleash Hidden Evidence.* Acessado em Novembro de 2016. Disponível em: <http://www.dataforensics.org/android-phone-forensics-analysis/>.

[XDADevelopers]XDADEVELOPERS. *ByPass Lockscreen.* Acessado em Novembro de 2016. Disponível em: <http://forum.xda-developers.com/galaxy-s4/general/bypass-t3035553>.

ANEXOS

I. PRÉ-REQUISITOS PARA AS ANÁLISES

As ferramentas apresentadas neste Anexo são indispensáveis para a realização da pesquisa.

I.1 Instalação Python 2.7

```
sudo apt-get install git
sudo apt-get install mesa-utils
```

Para instalar o Python 2.7, primeiramente, instale algumas dependências essenciais para seu funcionamento, com os seguintes comandos:

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-
dev tk-dev libgdbm-dev
```

É preciso que se faça o *download* da versão que se quer instalar, no *site* do Python. O formato indicado é a versão *Gzipped source tar ball*. Depois que o *download* for concluído, acesse a pasta em que foi salvo (se quiser um lugar específico, mova o arquivo para o lugar determinado) e use o seguindo comando:

```
tar -xvzf Python-x.tgz
```

Onde a letra 'x' ao lado de Python, é a versão específica que foi baixada. Depois disso, clique na pasta em que o arquivo tenha sido descompactado.

```
cd Python-x
```

Já na pasta execute os seguintes comandos para proceder a instalação:

```
./configure
make
```

I.2 Instalação Java 8

Instalando o Java 8 nas máquinas Linux:

- Abra um terminal

- Adicione o repositório do programa com o comando: sudo add-apt-repository ppa:openjdk-r/ppa
- Atualize o gerenciador de pacotes com o comando: sudo apt-get update
- Para instalar o programa, utilize o comando: sudo apt-get install openjdk-8-jdk
- Para conferir a versão que foi instalada, utilize o comando: java -version

É importante que algumas variáveis de ambiente sejam determinadas para o Java, da seguinte maneira:

```
JAVA_HOME=/usr/lib/jvm/jdk-x.x.x
export JAVA_HOME
```

I.3 Para compactar/descompactar arquivos

I.3.1 Se o arquivo for *.zip*

Instale a ferramenta da seguinte maneira:

```
sudo apt-get install unzip
```

Depois de instalada, se for preciso descompactar certo arquivo para uma pasta de destino particular, pode ser utilizado o seguinte:

```
sudo unzip file.zip -d destination\_folder
```

I.3.2 Se o arquivo for *tar.gz*

Quando um arquivo estiver neste tipo de formato, não é preciso instalar ferramenta extra alguma, pois, o próprio Linux já faz esta descompactação com o *tar* e o *gzip* da seguinte maneira:

```
tar -xvzf community\_images.tar.gz
```

O usuário pode acessar *man tar* para mais informações sobre os parâmetros a serem utilizados. Neste comando que foi exemplificado, as letras utilizadas com o *tar* significam:

- **f:** será sempre o último parâmetro do comando e o arquivo *tar* tem que ser posicionado logo depois deste. Serve para comunicar o *tar*, o nome e caminho do arquivo compactado;
- **z:** comunica o *tar* que ele deve descompactar, o arquivo usando *gzip*;
- **x:** o *tar* pode compactar ou descompactar arquivos. O parâmetro 'x' indica que é uma descompactação;
- **v:** tem como utilidade mostrar todos os arquivos que estão sendo extraídos.

I.4 Instalação Android Studio

Primeiramente, é preciso instalar as seguintes bibliotecas:

```
sudo apt-get install libc6:i386 libncurses:i386 libstdc++6:i386 lib32z1 libbbz2-
1.0:i386 lib64stdc++6:i386
```

Para instalar o Android Studio, primeiro é preciso fazer o *download* diretamente no seguinte endereço eletrônico [SDK]. Depois disso, descompacte o arquivo *.zip* em um lugar apropriado para suas aplicações, por exemplo, em */usr/local/* para uso no próprio perfil de usuário ou em */opt/* para usuários compartilhados.

Despois de descompactar, para executar o Android Studio, abra um terminal e vá até o diretório */android-studio/bin*, e execute *studio.sh*.

```
./studio.sh
```

Quando for executado, será questionado se você deseja importar configurações anteriores do Android Studio ou não e, em seguida, clique em OK. O assistente de configuração do Android Studio orienta o restante da configuração, que inclui o *download* de componentes do SDK do Android que são necessários para o desenvolvimento.

Depois de instalar o Android Studio, abra o diretório em que foi instalado o SDK (*/home/user/Android/Sdk/tools*) e execute *./android* para instalar dependências adicionais do SDK. No desenvolvimento deste trabalho, foram usados o NDK, CMake e LLDB como configurações extras do Android Studio.

I.5 Instalação *The Sleuth Kit* e *Autopsy*

É preciso obter as duas ferramentas em [SleuthKit] e [Autopsy]. E, em consequência, serão obtidos dois arquivos de extensão *.tar.gz*, que devem ser descompactados da seguinte maneira:

```
tar xzf sleuthkit-4.3.0.tar.gz
tar xzf autopsy-2.24.tar.gz
```

O TSK tem que ser instalado antes do *Autopsy*. Para isso, vá até o diretório do TSK e execute o arquivo de configuração e, depois, execute o comando *make*. Por último, faça um *sudo make install*. Estes passos estão descritos abaixo:

```
cd sleuthkit-4.3.0/
./configure
make
sudo make install
```

Assim, a instalação do TSK está finalizada. Para instalar o *Autopsy*, vá até a pasta em que foi instalado e execute o arquivo de configuração.

```
cd autopsy/
./configure
```

Ao executá-lo, aparecerá uma mensagem sobre o arquivo de configuração da biblioteca NSRL de *hashes* do NIST e deve-se informar a opção "n". Depois, será pedido para configurar o diretório que será utilizado para o *Evidence Locker*, onde serão salvos todas as informações do *Autopsy*; para isso, digite um caminho para ser criado.

Depois, ainda na pasta */autopsy-2.24*, execute *./autopsy* para começar a utilização da ferramenta.

I.6 Instalação LiME, *Volatility*, *Scalpel*, *Foremost* e *Heimdall*

Todas as ferramentas foram salvas em um diretório específico, o */usr/src/goldfish*, o mesmo foi criado para facilitar os testes e para que as informações coletadas fossem salvas em um mesmo diretório.

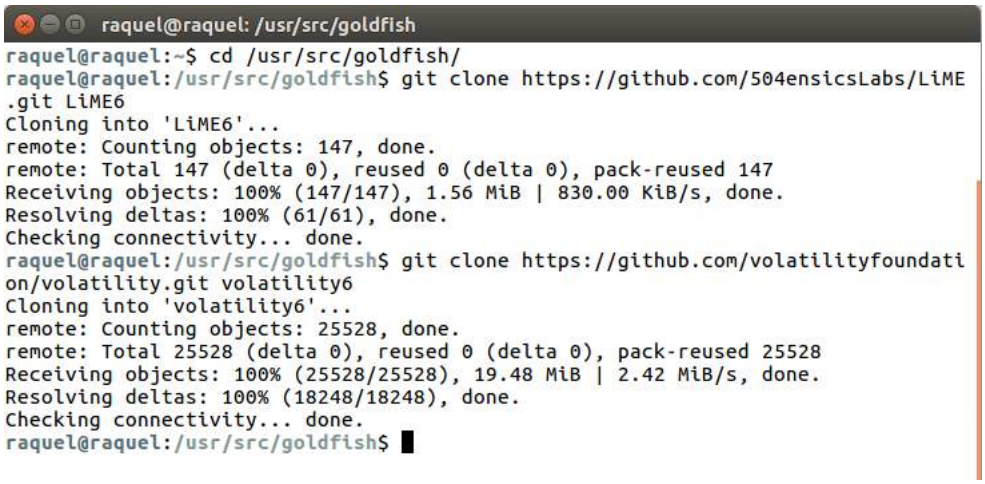


Figura I.1: Instrução de instalação das ferramentas LiME e *Volatility*

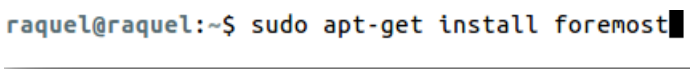


Figura I.2: Comando de instalação da ferramenta *Foremost*

```
raquel@raquel:~$ sudo apt-get install scalpel
[sudo] password for raquel:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  scalpel
0 upgraded, 1 newly installed, 0 to remove and 91 not upgraded.
Need to get 29,3 kB of archives.
After this operation, 112 kB of additional disk space will be used.
Get:1 http://br.archive.ubuntu.com/ubuntu xenial/universe amd64 scalpel amd64 1.
60-3 [29,3 kB]
Fetched 29,3 kB in 0s (228 kB/s)
Selecting previously unselected package scalpel.
(Reading database ... 398927 files and directories currently installed.)
Preparing to unpack .../scalpel_1.60-3_amd64.deb ...
Unpacking scalpel (1.60-3) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up scalpel (1.60-3) ...
raquel@raquel:~$ cd /etc/scalpel/
```

Figura I.3: Procedimento de instalação do *Scalpel*

```
raquel@raquel:~/Documents$ sudo apt-get install heimdall-flash
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  heimdall-flash
0 upgraded, 1 newly installed, 0 to remove and 92 not upgraded.
Need to get 41,7 kB of archives.
After this operation, 122 kB of additional disk space will be used.
Get:1 http://br.archive.ubuntu.com/ubuntu xenial/universe amd64 heimdall-flash
amd64 1.4.1-1 [41,7 kB]
Fetched 41,7 kB in 0s (230 kB/s)
Selecting previously unselected package heimdall-flash.
(Reading database ... 398935 files and directories currently installed.)
Preparing to unpack .../heimdall-flash_1.4.1-1_amd64.deb ...
Unpacking heimdall-flash (1.4.1-1) ...
Setting up heimdall-flash (1.4.1-1) ...
```

Figura I.4: Instalação da ferramenta *Heimdall*

I.7 Instalação do *Dwarfdump*

Para instalar essa dependência que será utilizada pela ferramenta *Volatility Framework*, use o seguinte comando:

```
sudo apt-get install dwarfdump
```

I.8 Instalação do *netcat* e *Busybox*

O procedimento foi utilizado na análise do primeiro cenário do Capítulo 4, Seção 4.2. A ferramenta foi obtida no seguinte endereço eletrônico [MobileResearch]. Depois disso, foi utilizada a função *push* do ADB para mover o arquivo para o dispositivo da seguinte maneira:

```
adb push old_nc /dev/Examiner/nc
```

Após este procedimento, deve-se dar a permissão de execução ao arquivo. É necessário acessar o *shell* do dispositivo e usar o seguinte comando:

```
chmod 777 /dev/Examiner/nc
```

Depois é só utilizar a ferramenta, como instruído no Capítulo 4.

A ferramenta *busybox*, utilizada no Capítulo 4, Seção 4.4, foi obtida no seguinte endereço eletrônico [Stephen]. Depois de ter feito o *download*, foi preciso utilizar a função *install* do ADB para executar o aplicativo diretamente no dispositivo.

```
adb install busybox-44.apk
```

I.9 Instalação *DB Browser for SQLite*, *Bless Hex Editor* e *Systemback*

Para a ferramenta *DB Browser for SQLite* deve-se adicionar o repositório da ferramenta, via linhas de comando no terminal:

```
sudo add-apt-repository -y ppa:linuxgndu/sqlitebrowser
```

Depois, faça um *update* no *cache* usando:

```
sudo apt-get update
```

E instale a ferramenta usando:

```
sudo apt-get install sqlitebrowser
```

Instale a ferramenta *Bless Hex Editor* usando:

```
sudo apt-get install bless
```

Instale a ferramenta *Systemback* usando:

```
sudo apt-add-repository ppa:nemh/systemback
```

```
sudo apt-get update
```

```
sudo apt-get install systemback
```

II. INSTRUÇÕES SOBRE O USO DO AVD E ADB

II.1 Android Debug Bridge - ABD

Serão apresentadas aqui, as principais funções do ADB que tem como utilidade o gerenciamento de dispositivos com sistema operacional Android.

- ***adb devices*:** verifica o estado da conexão dos dispositivos conectados;
- ***adb reboot recovery*:** reinicia o dispositivo no modo de recuperação (*recovery mode*);
- ***adb push [origem] [destino]*:** copia arquivos do seu computador para seu dispositivo;
- ***adb pull*:** copiar arquivos do dispositivo para o computador;
- ***adb install [arquivo.apk]*:** instala um aplicativo, a partir de um arquivo APK que está armazenado no seu computador;
- ***adb shell [comando]*:** executa comandos diretamente no sistema operacional do seu dispositivo Android;
- ***adb backup*:** cria um *backup* completo do seu aparelho e salva no computador;
- ***adb restore [arquivo_ backup.zip]*:** restaura o *backup* realizado pelo comando anterior;
- ***adb forward local remote*:** encaminha conexões de uma porta específica para uma outra no dispositivo.

Observação: Se mais de um dispositivo estiver conectado ao computador, para executar qualquer um dos comandos descritos acima, é preciso executá-lo da seguinte forma:

```
adb [-d|-e|-s serial_number] comando
```

O *serial_ number* é obtido após o uso de *adb devices* (único comando que não precisa de especificações de dispositivos).

II.2 Android Virtual Device - AVD

A criação de um AVD pode iniciar-se por linha de comando, em um terminal. Pode-se usar o comando *android avd* para criar os emuladores. Para iniciá-los, também no terminal, usa-se *emulator -avd emulator_ nome*.

III. EXEMPLO DE SIMULAÇÕES EM EMULADORES

Algumas simulações foram feitas para que evidências fossem produzidas nos emuladores para, só então, analisá-las. Estas serão mostradas a seguir.

III.1 Simular SMS's e ligações telefônicas

Estas simulações foram feitas utilizando-se o comando *Telnet* e, a seguir, alguns comandos e figuras representarão as mesmas. Para descobrir em qual porta o emulador está funcionando, pode-se utilizar o comando *adb devices* e, então, o mesmo retornará a informação. No caso apresentado agora, este valor foi 5556. Depois desta informação, pode-se comunicar com o emulador.

```
telnet localhost 5556
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: Authentication required
Android Console: type 'auth <auth_token>' to authenticate
Android Console: you can find your <auth_token> in
'/home/raquel/.emulator_console_auth_token'
OK
```

Neste caso, foi pedido uma informação para possibilitar a comunicação e essa está localizada em */home/raquel/.emulator_console_auth_token*. Utilizando-se o *gedit (gedit /home/raquel/.emulator_console_auth_token)*, a informação foi descoberta e pôde-se continuar com os testes.

```
auth syuOx3TebYlDUNoo
Android Console: type 'help' for a list of commands
OK
sms send 123456 Este é um teste de SMS para um emulador Android.
OK
gsm call 123456
OK
gsm accept 123456
OK
gsm call 123456
OK
```

```
gsm cancel 123456
OK
exit
Connection closed by foreign host.
```

Como visto, vários testes podem ser feitos e a Figura I.1 mostra como o emulador apresenta a ligação efetuada por meio do *Telnet*.

Figura III.1: Exemplo de como o emulador recebe a ligação efetuada como teste

III.2 Criar contatos na agenda

Criar um contato em um emulador é o mesmo que criar um contato em dispositivo real. No menu de aplicativos do emulador, vá em *People* e crie um contato indo até a opção *Create a new contact* e depois em *Keep local*. Crie quantos contatos quiser para fazer os testes necessários. A Figura I.2 mostra um exemplo de um contato.

Figura III.2: Exemplo de como criar um contato no emulador

Depois disso, pode-se continuar utilizando as ferramentas.

IV. ARQUIVOS UTILIZADOS NA COMPILAÇÃO DE FERRAMENTAS

IV.1 Arquivo *main.c* do LiME

```
/*
 * LiME - Linux Memory Extractor
 * Copyright (c) 2011-2014 Joe Sylve - 504ENSICS Labs
 *
 *
 * Author:
 * Joe Sylve       - joe.sylve@gmail.com, @jtsylve
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
#include ‘‘lime.h’’
// This file
static int write_lime_header(struct resource *);
static ssize_t write_padding(size_t);
static void write_range(struct resource *);
static ssize_t write_vaddr(void *, size_t);
static int setup(void);
static void cleanup(void);
static int init(void);
// External
extern int write_vaddr_tcp(void *, size_t);
extern int setup_tcp(void);
```

```
extern void cleanup_tcp(void);
extern int write_vaddr_disk(void *, size_t);
extern int setup_disk(void);
extern void cleanup_disk(void);
static char * format = 0;
static int mode = 0;
static int method = 0;
static char zero_page[PAGE_SIZE];
char * path = 0;
int dio = 0;
int port = 0;
int localhostonly = 0;
long timeout = 1000;
extern struct resource iomem_resource;
module_param(path, charp, S_IRUGO);
module_param(dio, int, S_IRUGO);
module_param(format, charp, S_IRUGO);
module_param(localhostonly, int, S_IRUGO);
module_param(timeout, long, S_IRUGO);
int init_module (void)
{
if(!path) {
DBG(“No path parameter specified”);
return -EINVAL;
}
if(!format) {
DBG(“No format parameter specified”);
return -EINVAL;
}
DBG(“Parameters”);
DBG(“  PATH: %s”, path);
DBG(“  DIO: %u”, dio);
DBG(“  FORMAT: %s”, format);
DBG(“  LOCALHOSTONLY: %u”, localhostonly);
memset(zero_page, 0, sizeof(zero_page));
if (!strcmp(format, “raw”)) mode = LIME_MODE_RAW;
else if (!strcmp(format, “lime”)) mode = LIME_MODE_LIME;
else if (!strcmp(format, “padded”)) mode = LIME_MODE_PADDED;
else {
DBG(“Invalid format parameter specified.”);
return -EINVAL;
}
```

```
method = (sscanf(path, ‘‘tcp:%d’’, &port) == 1) ? LIME_METHOD_TCP :
LIME_METHOD_DISK;
return init();
}
static int init() {
struct resource *p;
int err = 0;
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,18)
resource_size_t p_last = -1;
#else
__PTRDIFF_TYPE__ p_last = -1;
#endif
DBG(‘‘Initilizing Dump...’’);
if((err = setup())) {
DBG(‘‘Setup Error’’);
cleanup();
return err;
}
for (p = iomem_resource.child; p ; p = p->sibling) {
if (strcmp(p->name, LIME_RAMSTR))
continue;
if (mode == LIME_MODE_LIME && (err = write_lime_header(p))) {
DBG(‘‘Error writing header 0x%lx - 0x%lx’’, (long) p->start, (long) p->end);
break;
} else if (mode == LIME_MODE_PADDED && (err = write_padding((size_t)
((p->start - 1)
- p_last)))) {
DBG(‘‘Error writing padding 0x%lx - 0x%lx’’, (long) p_last, (long) p->start - 1);
break;
}
write_range(p);
        p_last = p->end;
}
DBG(‘‘Memory Dump Complete...’’);
cleanup();
return err;
}
static int write_lime_header(struct resource * res) {
ssize_t s;
lime_mem_range_header header;
memset(&header, 0, sizeof(lime_mem_range_header));
header.magic = LIME_MAGIC;
```

```
header.version = 1;
header.s_addr = res->start;
header.e_addr = res->end;
s = write_vaddr(&header, sizeof(lime_mem_range_header));
if (s != sizeof(lime_mem_range_header)) {
DBG(“Error sending header %ld”, s);
return (int) s;
}
return 0;
}
static ssize_t write_padding(size_t s) {
size_t i = 0;
ssize_t r;
while(s -= i) {
i = min((size_t) PAGE_SIZE, s);
r = write_vaddr(zero_page, i);
if (r != i) {
DBG(“Error sending zero page: %d”, r);
return r;
}
}
return 0;
}
static void write_range(struct resource * res) {
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,6,18)
resource_size_t i, is;
#else
__PTRDIFF_TYPE__ i, is;
#endif
struct page * p;
void * v;
ssize_t s;
for (i = res->start; i <= res->end; i += is) {
p = pfn_to_page((i) >> PAGE_SHIFT);
        is = min((size_t) PAGE_SIZE, (size_t) (res->end - i + 1));
        if (is < PAGE_SIZE) {
         // We can’t map partial pages and
         // the linux kernel doesn’t use them anyway
         DBG(“Padding partial page: vaddr %p size: %lu”, (void *) i, is);
         write_padding(is);
        } else {
v = kmap(p);
```

```
s = write_vaddr(v, is);
kunmap(p);
if (s != is) {
DBG(“Error writing page: vaddr %p ret: %d”, v, s);
return (int) s;
} else if (s != is) {
                     DBG(“Short Read %zu instead of %lu.  Null padding.”, s,
                     (unsigned long) is);
                    write_padding(is - s);
             }
}
}
return 0;
}
static int write_vaddr(void * v, size_t is) {
return (method == LIME_METHOD_TCP) ? write_vaddr_tcp(v, is) :
write_vaddr_disk(v, is);
}
static int setup(void) {
return (method == LIME_METHOD_TCP) ? setup_tcp() : setup_disk();
}
static void cleanup(void) {
return (method == LIME_METHOD_TCP) ? cleanup_tcp() : cleanup_disk();
}
void cleanup_module(void)
{
}
MODULE_LICENSE(“GPL”);
```

IV.2 Makefile da Ferramenta LiME

```
# LiME - Linux Memory Extractor
# Copyright (c) 2011-2014 Joe Sylve - 504ENSICS Labs#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or (at
# your option) any later version.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
```

```
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
# Author: Raquel Beatriz Silva do Nascimento

obj-m := lime.o
lime-objs := tcp.o disk.o main.o
KVER := $(shell uname -r)
PWD := $(shell pwd)
KDIR_GOLD := /usr/src/goldfish/goldfish4/
CCPATH := /home/raquel/Android/Sdk/ndk-bundle/toolchains/arm-linux-androideabi-
4.9/prebuilt/linux-x86_64/bin
default:
$(MAKE) ARCH=arm CROSS_COMPILE=$(CCPATH)/arm-linux-androideabi- EXTRA_CFLAGS=-fno-
pic -C $(KDIR_GOLD) M=$(PWD) modules
mv lime.ko lime-goldfish_4.ko
tidy:
rm -f *.o *.mod.c Module.symvers Module.markers modules.order \.*.o.cmd \.*.ko.
cmd \.*.o.d
rm -rf \.tmp_versions
clean:
make tidy
rm -f *.ko
```

IV.3 Makefile da Ferramenta *Volatility*

```
# Author: Raquel Beatriz Silva do Nascimento
obj-m += module.o
KDIR := /usr/src/goldfish/goldfish4
CCPATH := /home/raquel/Android/Sdk/ndk-bundle/toolchains/arm-linux-
androideabi-4.9/prebuilt/linux-x86_64/bin
-include version.mk
all: dwarf
dwarf: module.c
$(MAKE) ARCH=arm CROSS_COMPILE=$(CCPATH)/arm-linux-androideabi- -C $(KDIR) \
CONFIG_DEBUG_INFO=y M=$(PWD) modules
dwarfdump -di module.ko > module_4.dwarf
```