Universidade Federal do Amazonas
Instituto de Computação
Programa de Pós-Graduação em Informática

Carlos Alberto da Costa Ramos

Uma Análise de Segurança e Privacidade sobre as Bibliotecas de Cache para Plataforma Android

Manaus
Dezembro de 2015

Carlos Alberto da Costa Ramos

Uma Análise de Segurança e Privacidade sobre as Bibliotecas de Cache para Plataforma Android

Trabalho apresentado ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Prof. Dr. Eduardo Luzeiro Feitosa

Manaus
Dezembro de 2015

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Ramos, Carlos Alberto da Costa
R175u Uma Análise de Segurança e Privacidade sobre as Bibliotecas de Cache para Plataforma Android / Carlos Alberto da Costa Ramos. 2015
86 f.: il. color; 31 cm.

Orientador: Eduardo Luzeiro Feitosa
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Cache. 2. Biblioteca. 3. Android. 4. Forense. 5. Modelo. I. Feitosa, Eduardo Luzeiro II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Uma Análise de Segurança e Privacidade sobre as Bibliotecas de Cache para Plataforma Android"

CARLOS ALBERTO DA COSTA RAMOS

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Eduardo Luzeiro Feitosa
Prof. Eduardo Luzeiro Feitosa - PRESIDENTE

Arilo Claudio Dias Neto
Prof. Arilo Claudio Dias Neto - MEMBRO INTERNO

Prof. José Luiz de Souza Pio - MEMBRO EXTERNO

Manaus, 10 de Dezembro de 2015

Agradecimentos

Primeiramente agradeço a Deus pela concessão de todas as coisas que nos cercam, providenciando todo um caminho e escolhendo as pessoas certas para fazerem parte de um propósito.

A oportunidade de ter sido escolhido na seleção de mestrado pelo meu Orientador professor Feitosa, pois não fazia ideia o que era um mestrado e tampouco que poderia ter entrado em contato antes da seleção para aprofundar um tema a ser trabalhado. Acabei entrando sem esse contato e considerei a oportunidade ímpar por ter sido escolhido, restando-me agradecer de forma especial sua disponibilidade, atenção dispensada, paciência, dedicação e seu profissionalismo. Além de orientador é um ser humano incrível, soube compreender algumas necessidades a qual estava passando. Acredito veementemente em Deus por ter lhe colocado neste propósito, pois só ele conhece o coração dos homens. Muito obrigado, de coração.

A todos os professores da UFAM, onde tive a oportunidade de abstrair conhecimentos a um nível excelente em detrimento a qualidade e desempenho aplicado às disciplinas ministradas. Em especial quero agradecer aos professores Arilo, Tayana e Nakamura. Foi uma honra tê-los conhecido. A professora Tayana e Arilo pela compreensão quando mais precisei de ajuda. Ambos foram "show de bola", desculpem o termo, mas expressa a gratidão e o entusiasmo de terem me ajudado sem pestanejar, sem este esforço não teria conseguido. Muito obrigado por terem sido meus mestres nesta jornada. Ao professor Nakamura, que não deixou que a turma desanimasse diante do desespero das notas, sempre conversando e mostrando que tudo tem seu valor, basta correr atrás e na hora do desespero "saber segurar o bicho pelo chifre e sustentar" (palavras do próprio professor). Aprendi o quanto podemos ser fortes e não deixar que adversidades nos deixem de fora daquilo que um dia sonhamos ser, não é as pessoas que ditam como será seu futuro, pois o futuro não existe, quem dita seu futuro é o que você faz no presente!

Ao Frank, Elienai, Ana e os demais que trabalham na secretaria do ICOMP pela paciência e profissionalismo no atendimento. Em especial deixo um grande abraço ao Frank, um grande profissional, possuidor de uma competência quilo-

métrica. Muito obrigado!

A minha esposa Ana Cristina pelo incentivo, compreensão e encorajamento, durante todo este período. Sei que não foi fácil suportar a distância, mas deu tudo certo e brevemente estaremos juntinhos novamente.

Aos meus amigos do mestrado, em especial aqueles que fizeram parte da minha vida neste período, que estavam sempre dispostos a ajudar e incentivar naquilo que fosse preciso e aos momentos de entusiasmo partilhados em conjunto. Aos amigos Caio, Fábio, Pablo, Adriana, Rayol, Thaís, Michel (japa), Haline, Ádria, Clahil, Bernardo, Ivan, Weslen, Thiago, Alex, Maiara, Maria, Adeilson, Hendrio, Kevin, Luis Sérgio, Isomar, Oziel, Delano, Janaína e Marcos. Espero não ter esquecido ninguém. Vocês foram demais! Desejo muito sucesso a todos.

Ao discente de graduação, futuro mestrando Guibson pela dedicação e a ajuda com o trabalho, abdicando de suas horas de lazer para ajudar nos testes desta pesquisa. Desejo sucesso, felicidades e muita saúde. Que seu futuro espelhe o que almeja. Um grande abraço!

A todos os demais ... obrigado!!!

Resumo

Esta dissertação apresenta uma investigação sobre bibliotecas que implementam cache de aplicação na plataforma Android, por meio da definição de um modelo forense de análise e experimentação prática, visando comprovar o mau uso e/ou descaso no âmbito da segurança da informação, especialmente a informações sensíveis e/ou confidenciais, tendo como argumentos que os dados sensíveis precisam ser protegidos (com criptografia, por exemplo), bem como avaliar se as bibliotecas usam ou não, mecanismos de segurança e ainda saber se existem pesquisas sobre padronização de esquemas de segurança para Cache. Isso ocorre porque, embora existam algumas recomendações básicas de segurança, muitos desenvolvedores de aplicativos não as seguem ou deliberadamente criam mecanismos para capturar tais informações. Na vasta gama de soluções elaboradas para tratar do vazamento de dados sensíveis em dispositivos móveis, especialmente para plataforma Android, uma área pouco explorada é a análise dos dados de Cache das aplicações. Neste sentido, esta dissertação tem como objetivo analisar bibliotecas que implementam cache de aplicação na plataforma Android, através da definição de um modelo forense de análise e experimentação prática, visando comprovar o mau uso e/ou descaso no âmbito da segurança da informação, especialmente a informações sensíveis e/ou confidenciais.

Palavras-chave: Cache, Biblioteca, Android, Forense, Modelo, Vazamento, Dados.

Abstract

This work presents an investigation about Android libraries employed to implement cache in Android applications, through the definition of a forensic analysis model and practical experimentation, aiming to prove misuse and/or negligence in the context of information security. The ideia is to verify if the libraries use or not security mechanisms and follow some kind of security schemes for Cache In this context, this work aims to analyze libraries that implement application cache on the Android platform, by defining a forensic analysis model and practical experimentation, aiming to prove the misuse and/or negligence in the context of information security, especially information sensitive and/or confidential.

Keywords: Cache, Library, Android, Forensic, Model, Leak, Data.

Sumário

Capítulo 1

Introdução

Os inúmeros avanços tecnológicos dos últimos anos permitiram o crescimento do número de usuários conectados ao redor do mundo, especialmente daqueles usando dispositivos móveis.

Prova disso é o grande número de empresas que converteram suas aplicações tradicionais para plataformas móveis ou apenas lançaram aplicações para essa nova plataforma [1]. O fato é que o uso de dispositivos móveis tem facilitado a execução das mais variadas atividades diárias dos seres humanos.

Entretanto, toda essa popularidade fez dos dispositivos móveis alvos preferidos para atividades maliciosas. Spam [2, 3], códigos maliciosos [4], *phishing* [5, 6] e a participação em botnets [7, 8] são alguns exemplos de atividades ilícitas e não desejadas já "consagradas" no mundo móvel. Mais recentemente, o vazamento de informação, definido como a distribuição acidental ou não intencional de dados particulares ou confidenciais a uma entidade não autorizada [9], vem despontando e trazendo sérios problemas para empresas e usuários. Dentre as informações "vazadas" destacam-se, além de dados pessoais, o número do celular, o número de Identificação Internacional de Equipamento (IMEI - *International Mobile Equipment Identity*), contas de e-mail, entradas de cache do teclado, histórico do navegador, localização (GPS), entre outros [10].

Parte do problema do vazamento de informações reside no fato de que os dados transmitidos são, em grande parte, não regulamentados e não monitorados em seu caminho até seus destinos [11]. Por exemplo, na plataforma Android, os desenvolvedores são os responsáveis por garantir a segurança dos dados manipulados ou gerados por suas aplicações. Outra parte do problema é que empresas e desenvolvedores de aplicativos, na vontade de entregar ao mercado novos serviços, esquecem de empregar práticas seguras de programação ou seguir as recomendações da plataforma [12]. Um bom exemplo é a utilização da Cache[1] nas aplicações,

[1]O termo Cache pode ser compreendido como uma área de armazenamento onde dados ou

cuja finalidade é melhorar o desempenho e evitar novas conexões para refazer a busca por informações. Contudo, na plataforma Android, os desenvolvedores são apenas orientados (não obrigados) a limitar o tamanho da Cache em 1MB [13] e, se possível, salvar os dados em Cache de modo cifrado.

É neste contexto de falta de recomendações que esta dissertação se enquadra ao tentar identificar possíveis problemas de segurança através da avaliação da Cache em aplicativos Android.

1.1 Motivação

Para entender a motivação para a realização da pesquisa nesta dissertação é preciso entender o que os usuários entendem por segurança no mundo móvel. O relatório do *Internet Security Threat Report* (ISTR20) de 2015, da Symantec [14], e a pesquisa *Norton Mobile* [15] de 2014 demonstram que muitos consumidores do mundo móvel, embora preocupados com a segurança de aparelhos e dados, estão dispostos a permitir que aplicativos acessem suas informações pessoais, como senhas, contatos, geolocalização. Além disso, a pesquisa Norton Mobile [15] aponta que no Brasil 74% dos usuários de smartphones disponibilizam dados e informações pessoais aos desenvolvedores de aplicativos móveis e 89% dos brasileiros se preocupam com vírus e malware, porém entendem pouco sobre segurança móvel.

O fato é que grande parte dos usuários de aplicativos móveis desconhece que abrem mão da privacidade e que fornecem informações e dados pessoais aos desenvolvedores de aplicativos. Além disso, acreditam que entendem os termos do acordo quando baixam uma aplicação (app), mas na verdade conhecem muito pouco sobre as práticas e comportamentos comuns das permissões dadas a aplicativos.

A questão é: Como as plataformas móveis lidam com a segurança e a falta de preparo dos usuários? Segundo Braga et al. [16], os desenvolvedores de aplicativos iOS utilizam uma assinatura de código gerada pela empresa Apple[2], ou seja, um usuário da loja de aplicativos da Apple (App Store) só pode fazer download de aplicativos que foram analisados e posteriormente disponibilizado pela empresa.

Já na plataforma Android, a maior no mercado de dispositivos móveis [17] e foco desta dissertação, são empregados alguns mecanismos para tentar impedir o acesso a recursos restritos, e assim evitar ou amenizar problemas de segurança. Em linhas gerais, sempre que um aplicativo precisa acessar um determinado recurso, o desenvolvedor tem que pedir autorização do usuário para utilizá-lo,

processos frequentemente utilizados são guardados para um acesso futuro mais rápido, poupando tempo e uso desnecessário de algum recurso (hardware, por exemplo).

[2]http://www.apple.com/

declarando-o no arquivo de manifesto do aplicativo [18]. Desta forma, o Android avisa ao usuário que o aplicativo requer certos recursos restritos (por exemplo, dados de localização) e que ao instalar o aplicativo estará concedendo permissão de acesso a esses recursos. Se o usuário recusar a autorizar essas permissões, o aplicativo não será instalado. No entanto, também é preciso notar que ao aceitar as permissões, o usuário não será informado como e quando o recurso será utilizado.

Embora a plataforma Android empregue outros mecanismos e esquemas de segurança (*Bootloader/Recovery*, ADB e *Application Framework*)[19], certos aspectos são negligenciados. Um desses é o uso de Cache. Uma simples busca na loja de aplicativos do Google (Google Play) revela a existência de uma gama de aplicativos disponíveis com a finalidade de realizar a limpeza de dados existentes em Cache das aplicações, visando espaço de armazenamento, otimização de memória e ganhos de desempenho. Contudo, nenhuma delas leva em consideração (avalia) a segurança do usuário, justamente o item no qual se pode dar uma grande ênfase, uma vez que na Cache podem ser encontrados muitos dados (gerados pelas aplicações e possivelmente sensíveis) sem os devidos cuidados de segurança e que podem ser vazados.

Podjarny [20] publicou em 2011 um estudo no qual apresenta um comparativo sobre o tamanho da Cache de alguns navegadores, especialmente em dispositivos móveis. Os resultados mostram que a Cache em navegadores de dispositivos móveis é de fato pequena (até 5MB) enquanto que em computadores varia entre 5 a 150MB. O estudo gera um questionamento no sentido de analisar como as aplicações estão fazendo uso da Cache nos dispositivos móveis, levantando questões sobre as diretrizes de segurança (se os dados salvos em Cache estão disponíveis ou podem ser vazados por meio de aplicativos mal intencionados), tamanho (se a quantidade de dados em Cache não está ultrapassando uma quantidade recomendada), tempo médio (duração de vida do arquivo disponível), bem como outros pontos.

Partindo desta constatação, alguns questionamentos sobre o uso da Cache em Android podem ser feitos:

- As bibliotecas que implementam ou reimplementam Cache para as aplicações seguem algum tipo de regra, padrão ou práticas de uso?
- Existem normas da plataforma aplicadas no desenvolvimento e uso de Caches?
- Existem métricas que permitem avaliar o nível de segurança das aplicações que usam Cache?
- Existe algum modelo para a análise de Cache de dispositivos móveis?

Para resumir, torna-se claro que existe a necessidade de analisar os tipos de dados e informações disponíveis em Cache sob o foco de tais questionamentos, para então descrever se há ou não a possibilidade de ocorrer o vazamento de informação.

1.2 Objetivo

O objetivo desta dissertação é analisar bibliotecas que implementam cache de aplicação na plataforma Android, através da definição de um modelo forense de análise e experimentação prática, visando comprovar o mau uso e/ou descaso no âmbito da segurança da informação, especialmente a informações sensíveis e/ou confidenciais.

Para alcançar este objetivo, primeiramente será proposto um modelo de análise de Cache em dispositivos móveis, empregando o processo de investigação forense, a fim de permitir analisar aplicações Android e identificar a existência de dados sensíveis dos usuários armazenados em Cache. Em seguida, serão definidos fatores de avaliação, empregando conceitos das áreas de segurança da informação e arquitetura de computadores, de forma a permitir uma avaliação de bibliotecas e APIs que implementam Cache na plataforma Android. Por fim, será implementado um protótipo funcional (aplicação Android) para auxiliar na avaliação de bibliotecas de Cache com base nas métricas elaboradas.

1.3 Contribuições

A partir do desenvolvimento dos objetivos definidos nesta dissertação, é possível enumerar as seguintes contribuições:

1. Fatores de avaliação que permitem avaliar bibliotecas que implementam Cache. É importante salientar que tais fatores são genéricos o suficiente a ponto de serem utilizadas na avaliação de aplicativos móveis, *Web Caching* e uso geral de Cache.

2. Processo conceitual (modelo) de análise de Cache em plataforma Android, através da adaptação de um modelo para análise forense que atende a aspectos de Cache.

1.4 Estrutura do Documento

Este trabalho está organizado da seguinte forma. O Capítulo 2 está dividido em três partes. A primeira parte descreve os principais conceitos sobre a plataforma

Android. A segunda sobre as bibliotecas de Cache. E a última sobre os modelos forense. No capítulo 3 são discutidos os trabalhos relacionados. O Capítulo 4 apresenta o arcabouço proposto e implementado para análise de Caches, onde primeiramente é proposto um modelo forense para Cache e sua aplicabilidade na prática como forma de avaliação, num segundo momento é apresentado os fatores para avaliação das implementações de cache, em terceiro uma análise teórica (empírica) sobre as bibliotecas de Cache e por último, experimentos são desenvolvidos a partir de um protótipo, cuja análise dos resultados são realizadas baseadas no modelo forense proposto. O Capítulo 5 apresenta recomendações de segurança baseadas nas principais empresas de desenvolvimento de ferramentas e/ou aplicativos, bem como organizações de segurança da informação. No Capítulo 6 são feitas as considerações finais, as dificuldades encontradas, bem como são descritos os trabalhos futuros.

Capítulo 2

Conceitos Básicos

Este Capítulo apresenta os principais conceitos básicos necessários para a compreensão dos temas abordados. Primeiro, a plataforma Android é apresentada, incluindo pontos como arquitetura, modelo de segurança, estrutura dos diretórios e a organização da cache. Em seguida, as bibliotecas utilizadas para implementação de Cache na plataforma Android são apresentadas e uma descrição de suas características é feita. Por último, uma explicação sobre modelos para análise forense digital.

2.1 Plataforma Android

O Android é uma plataforma completa de software para dispositivos móveis, de código aberto e baseada no kernel Linux 2.6, composta por um sistema operacional, um *middleware* e aplicações. Gerido pela Open Handset Alliance[1], a plataforma Android teve e continua tendo um impacto significativo sobre o mercado de dispositivos móveis, em especial smartphones. Tal fato é comprovado pela estatística apresentada pelo grupo Gartner [17] sobre a participação Android no segundo trimestre de 2015 no mercado americano (Tabela 2.1).

Além das estatísticas, o predomínio do Android é facilmente justificado pela altíssima quantidade de produtos lançados (fabricantes como LG, Sony, Samsung, Motorola e HTC apresentam recorrentemente novos modelos).

2.1.1 Arquitetura do Android

O Google se refere ao Android como uma pilha de software, onde cada camada da pilha agrupa vários programas que suportam funções específicas do sistema

[1]Um grupo de 84 empresas de tecnologia móvel, unidas para acelerar inovações com padrões abertos. http://www.openhandsetalliance.com

Tabela 2.1: Vendas de smartphones no mundo a usuários finais por Sistema Operacional no 2° Trimestre 2015 - 2T15 (milhões de unidades). Fonte: [17]

Sistema Operacional	**Unidades (2T15)**	**Market Share (2T15)**	**Unidades (2T14)**	**Market Share (2T14)**
Android	271.010	82,2	243.484	83,8
iOS	48.086	14,6	35.345	12,2
Windows	8.198	2,5	8.095	2,8
BlackBerry	1.1153	0,3	2.044	0,7
Other OS	1.229	0,4	1.416,8	0,5
Total	**329.676,4**	**100**	**290.384,4**	**100**

operacional, conforme ilustrado na Figura 2.1.

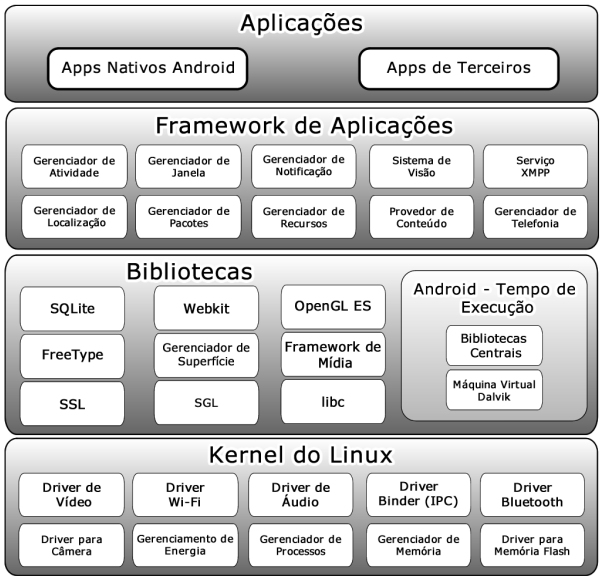

Figura 2.1: Arquitetura Android. Fonte: [21]

A base da pilha é o Kernel Linux, versão 2.6, que inclui funções de baixo nível como programas de gerenciamento de memória, configurações de segurança, software de gerenciamento de energia e vários drivers (programas que controlam dispositivos de hardware). Por exemplo, o kernel do Android inclui um driver de câmera para permitir ao usuário enviar comandos (tirar fotos ou gravar vídeos, por exemplo) ao hardware da câmera.

O próximo nível da pilha inclui as bibliotecas do Android, que fornecem funcionalidades básicas e essenciais para os desenvolvedores, como *WebKit* para renderização HTML para aplicativos próprios ou de terceiros. As bibliotecas

centrais executam de forma personalizada junto com a Máquina Virtual Java (*Java Virtual Machine - JVM*), proporcionando o ambiente de tempo de execução do Android onde os aplicativos são executados. O acesso a estes recursos se dá por meio de APIs (*Application Program Interface*) disponibilizados via SDK (*Software Development Kit*) ou NDK (*Native Development Kit*).

Neste mesmo nível existe a camada de tempo de execução do Android (*Android Runtime*), que inclui um conjunto de bibliotecas essenciais Java (*Android core*) - fornecem APIs primárias para desenvolvedores escreverem aplicativos Android - bem como a Máquina Virtual Dalvik (*Dalvik Virtual Machine* - DVM). De acordo como [22], as bibliotecas *Android core* se dividem em três categorias principais:

- **Bibliotecas da VM Dalvik** - Conjunto de bibliotecas usadas predominantemente para interagir diretamente com uma instância da VM Dalvik;
- **Bibliotecas de interoperabilidade Java** - Uma vez que os aplicativos do Android são desenvolvidos utilizando a linguagem Java, essas bibliotecas fornecem suporte para tarefas como, por exemplo, manipulação de strings, *networking* e manipulação de arquivos. As bibliotecas de Interoperabilidade Java são uma implementação *open source* (baseada no projeto Apache Harmony) de um subconjunto das bibliotecas centrais Java padrão que foram adaptadas e transformadas para uso por aplicativos executados em uma VM Dalvik.
- **Bibliotecas Android** - Esta categoria abrange as bibliotecas baseadas em Java que são específicas para o desenvolvimento do Android. Exemplos de bibliotecas nesta categoria incluem as bibliotecas do framework de aplicação.

O outro item da *Android Runtime* é a **Dalvik Virtual Machine (DVM)**, um componente essencial e de diferença da plataforma Android. A VM Dalvik é otimizada para os requisitos de pouca memória e foi projetada para permitir múltiplas instâncias (várias VMs) para executar simultaneamente [23]. Uma vez que foi desenvolvida para criar um ambiente eficiente e seguro, cada aplicativo é executado em sua própria VM [21]. Isso garante que: (i) nenhuma aplicação é dependente de outra; (ii) se uma aplicação parar, ela não afeta quaisquer outras aplicações executando no dispositivo; (iii) o gerenciamento de memória é mais simples. A Figura 2.2 exemplifica como se dá a execução em cada Máquina Virtual.

Embora as aplicações Android sejam em Java, a VM Dalvik é diferente do padrão da VM Java em alguns aspectos [23]. Em primeiro lugar, a maioria das VMs utiliza uma arquitetura baseada em pilha, mas Dalvik é uma arquitetura

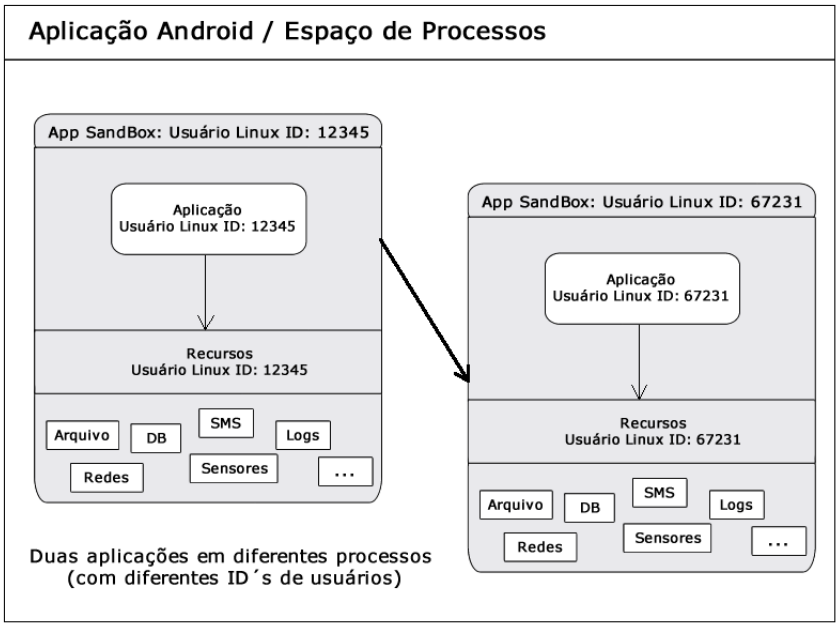

Figura 2.2: Dois aplicativos Android, cada um em seu próprio ambiente de simulação ou processo básico. Fonte: [24]

baseada em registradores. Em segundo lugar, Dalvik executa aplicativos Java que foram transformados para o formato Dalvik Executable (DEX). Em linhas gerais, a VM Dalvik é uma forma de gerar segurança na execução das aplicações, buscando garantir que os dados de uma aplicação não possam ser acessados por outra aplicação. Partindo desta ideia, o Google possui **modelos de segurança** para quem desenvolve aplicações para Android (maiores detalhes na Seção 2.1.2).

A próxima camada da pilha é o framework de aplicação. Ele inclui os programas que gerenciam as funções básicas do dispositivo, como alocação de recursos, aplicações de telefone, mudança entre processos ou programas, entre outros. Os desenvolvedores de aplicações têm acesso total ao framework de aplicações do Android. Isso possibilita que eles tirem vantagem das capacidades de processamento do Android e suportem recursos quando estão construindo uma aplicação Android. Em linhas gerais, o framework de aplicações pode ser visto como um conjunto de ferramentas básicas com o qual um desenvolvedor pode construir ferramentas muito mais complexas.

Por fim, no topo da pilha estão as aplicações propriamente ditas.

2.1.2 Modelo de Segurança do Android

No livro *"Android Forensics: Investigation, Analysis and Mobile Security for Google Android"*, Hoog [21] afirma que a plataforma Android implementa uma série de controles de segurança para proteger o usuário. Contudo, todos eles são realizados apenas durante o processo de instalação do aplicativo no dispositivo.

O primeiro desses controles é a **verificação da assinatura digital do desenvolvedor**. Por padrão, todos os aplicativos Android (arquivos *.apk*) devem ser assinados. Assim, ao instalar um aplicativo, o Android analisa o arquivo *.apk* para garantir que existe uma assinatura digital válida que identifique o desenvolvedor. Além de identificar o autor do código, a assinatura serve para detectar se o aplicativo foi alterado e para estabelecer a confiança entre aplicativos para que possam compartilhar código e dados de forma segura.

Contudo, a assinatura digital exigida para o desenvolvedor não precisa ser gerada (assinada) por uma entidade certificadora (*Certificate Authority* - CA) real, ou seja, pode ser auto-assinada. Assim, é responsabilidade do desenvolvedor gerar e manter as chaves seguras. Um desenvolvedor pode inscrever mais de um aplicativo com a mesma certificação digital, mas Hoog [21] esclarece que esta situação é "excepcional", comumente usada quando um desenvolvedor tem tanto uma versão gratuita quanto uma versão paga do mesmo aplicativo. Desta forma, quando o usuário comprar o produto após ter usado a versão gratuita, poderá aproveitar dos dados já existentes no dispositivo.

O segundo controle, após a validação do arquivo *.apk*, é a **verificação de permissões**. No Android, o acesso a partes relacionadas à privacidade do usuário e relevantes para a segurança é controlado por um sistema de instalação em tempo real de permissão para aplicações.

Em outras palavras, qualquer aplicativo Android não tem, por padrão, permissões associadas a ele, o que significa que não pode fazer nada que possa afetar negativamente a experiência do usuário ou todos os dados do dispositivo. Para fazer uso de recursos do dispositivo, o desenvolvedor do aplicativo deve incluir em arquivo especial, chamado de *AndroidManifest.xml*, com uma ou mais permissões (no formato *<user-permission>*) que definem/declaram as permissões necessárias para o funcionamento da aplicação. Se o usuário não quiser conceder uma permissão para o aplicativo, ele pode cancelar o processo de instalação.

Por exemplo, se uma aplicação de troca de mensagens precisa de acesso aos contatos do usuário, mensagens SMS e da rede/Internet, ela expressa tais permissões no arquivo de manifesto e usuário provavelmente deve aceitá-las. No entanto, se essas mesmas permissões forem solicitadas por uma aplicação tipo calculadora, o usuário pode e deve questionar a autorização e escolher não instalar o aplicativo.

Para melhor ilustrar esse processo, a Figura 2.3 apresenta, a esquerda, a

interface de solicitação de permissão do aplicativo FourSquare e, a direita, um trecho do conteúdo do arquivo *AndroidManifest.xml* dessa aplicação.

(a) Interface com permissões

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   android:installLocation="auto" package="com.joelapenna.foursquared"
   platformBuildVersionCode="21"
   platformBuildVersionName="5.0-1521886">
   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
   <uses-permission android:name="android.permission.INTERNET"/>
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
   <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
   <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
   <uses-permission android:name="android.permission.VIBRATE"/>
   <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
   <uses-permission android:name="android.permission.WAKE_LOCK"/>
   <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
   <uses-permission android:name="android.permission.NFC"/>
   <uses-permission android:name="android.permission.READ_CONTACTS"/>
   <uses-feature android:glEsVersion="0x20000" android:required="true"/>
   <meta-data android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version"/>
</manifest>
```

(b) Trecho do *AndroidManifest.xml*

Figura 2.3: Permissões para instalação do aplicativo Foursquare

Vale ressaltar que na plataforma Android não existe um mecanismo de checagem das permissões dadas aos aplicativos. Uma boa argumentação sobre esse problema é feita por Felt *et al.* [25] ao afirmar que a validação das permissões em tempo de instalação pode fornecer aos usuários o controle sobre sua privacidade e reduzir o impacto de bugs e vulnerabilidades em aplicações.

No entanto, em tempo de instalação, o sistema de checagem de permissão é ineficaz se os desenvolvedores rotineiramente solicitam mais permissões do que eles exigem. Aplicações sobre-privilegiadas expõem os usuários a avisos de permissão desnecessários e aumentam o impacto de um bug ou vulnerabilidade. Soma-se a isso o fato de que, em sua maioria, os usuários acabam consentindo, de forma rápida, as permissões e solicitações dos aplicativos.

O terceiro e último controle relatado por Hoog [21] são as **identificações de usuário - user ID**. Por padrão, no Android cada aplicativo ao ser instalado tem uma identificação de usuário (*UID*), padrão Linux, atribuída a ele. Uma vez atribuído, um UID permanece constante e único para aquela aplicação durante o tempo de vida do dispositivo. Obviamente, em um dispositivo diferente, a mesma aplicação pode ter um UID diferente.

Além do UID, cada aplicativo também tem um identificador de grupo (**group ID**) atribuído e, como explicado na Seção 2.1.1, cada aplicativo é executado como um único processo da VM Dalvik. Essas duas identificações (UID e GID) garantem que as aplicações não compartilham memória, permissões ou armazenamento em disco.

Uma vez cumprido esses três controles, a aplicação é instalada no dispositivo e sua estrutura de pastas (diretórios) é criada.

2.1.3 Estrutura de Diretórios

Uma vez cumprido os três controles de segurança da seção anterior, cada aplicação instalada em um dispositivo Android terá sua estrutura de diretórios criada. Para melhorar o entendimento da estrutura de diretórios do Android, a Figura 2.4 ilustra essa estrutura, obtida do comando *tree*.

No topo da estrutura de diretórios do Android está a raiz (**Linha 1**), que cria a estrutura e aponta para os outros sistemas de arquivos. **Linha 2 a 5**: Diretório */app-cache*, um diretório temporário onde se vê a estrutura de Cache do navegador presumivelmente e onde, ao longo do tempo, outras aplicações poderão fazer uso deste espaço.

Nas **Linhas 6 a 8** encontra-se o diretório dedicado a */cache* onde comumente são encontrados arquivos de visualização do Gmail, dados do navegador, alguns downloads, bem como atualizações de algumas operadoras de telefonia. A **Linha 9** representa a raiz do diretório *"/data"* que contém diversos subdiretórios. A **Linha 10** ilustra o diretório *"/data/app"* que contém os arquivos *.apk* da Play Store. O diretório *"/data/dalvik-cache"* (**Linha 15**) contém os arquivos *dex* em cache da VM Dalvik usados para executar aplicativos. A **Linha 16** mostra o diretório *"/data/data"*, que contém os dados específicos da aplicação, facilmente a área mais importante para focar em uma investigação. Das **Linhas 17 a 23** é representado, para fins de demonstração, a hierarquia de diretórios de uma aplicação (app). O diretório é nomeado de acordo com o nome do pacote e, muitas vezes identifica claramente o desenvolvedor (Facebook, neste caso).

O diretório *"/data/local"* (**Linha 25**) é importante, pois permite um shell com acesso de leitura/gravação. Quando um aplicativo é instalado, ele é copiado primeiro para *"/data/local"*. Além disso, algumas técnicas forenses contam com este diretório para fazer upload de arquivos importantes, geralmente binários. Para finalizar, a pasta "*/data*"(**Linha 36**) representa o diretório *"/data/property"* que contém várias propriedades do sistema tais como fuso horário, país e idioma.

O diretório *"/mnt"* (**Linha 44**) é onde o sistema monta vários sistemas de arquivos, incluindo cartão SD, eMMC, entre outros.

O diretório *"/system"* (**Linha 74**) contém vários subdiretórios. A **Linha 75** representa o diretório *"/system/app"* e contém arquivos *.apk* de aplicativos fornecidos com o sistema. Isso inclui aplicativos agrupados por Google/Android, do fabricante e da operadora de telefonia móvel. Já o diretório *"/system/bin"* (**Linhas 76**) contém os arquivos binários do Android utilizados no sistema. Analistas forenses e engenheiros de segurança (e a maioria dos pesquisadores Android)

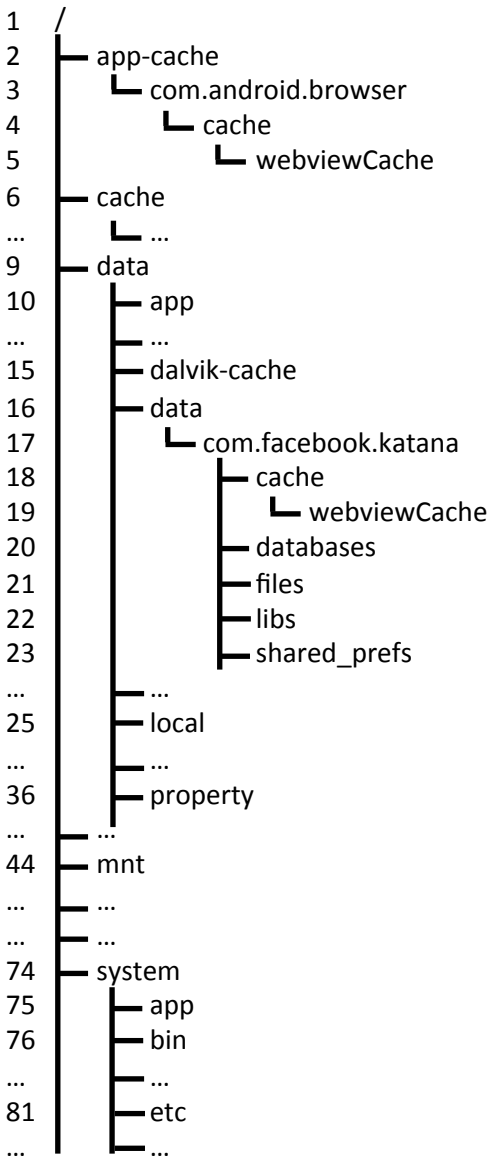

Figura 2.4: Estrutura de Diretórios do Android. Fonte: [21]

podem encontrar diversos comandos úteis e não documentados através da experimentação de arquivos nesses diretórios. Por fim, a **Linha 81** mostra o diretório *"/system/etc"*, local onde a Play Store armazena as configurações típicas Linux/Unix. Ele contém vários arquivos de configuração, podendo variar em outros dispositivos.

2.1.3.1 Armazenamento

No que diz respeito ao armazenamento, na página Web para desenvolvedores Android [9] descrevem-se três maneiras para salvar os dados no dispositivo: (1) usando armazenamento interno; (2) armazenamento externo e (3) provedores de conteúdo. Como destacado por Hoog [21], nas áreas de armazenamento de dados externos *(cartão SD e cartões SD emulados)* as aplicações podem armazenar dados em qualquer local que desejar. No entanto, o armazenamento de dados interno é controlado pelas APIs Android, sendo sempre empregado o subdiretório das aplicações "*/data/data*".

A Tabela 2.2 referencia algumas áreas de armazenamento mais comuns.

Tabela 2.2: Fonte: [21]

Diretório	**Descrição/Funcionalidade**
shared_prefs	Diretório de armazenamento de preferências compartilhadas em formato XML
lib	Arquivos de biblioteca personalizado requerido por aplicativos.
files	Arquivos que o desenvolvedor salva no armazenamento interno
cache	Arquivos de cache dos aplicativos (Cache do navegador da Web ou outros aplicativos que usam o motor WebKit)
databases	Bancos de dados SQLite e arquivos do tipo journal

2.1.4 Organização da Cache

A Figura 2.5 mostra que a Cache está dividida em: (i) Sistema, que contém o Cache da máquina virtual Dalvik; e (ii) Aplicações (app), onde o armazenamento ocorre na memória do dispositivo (Cache interna) e/ou no cartão de memória externo (Cache externa).

A Cache de **Sistema** representa a Cache da Dalvik VM. Seu propósito é permitir um tempo de execução mais rápido através do alinhamento, verificação e otimização de *byte-codes*, feitos com antecedência e armazenados como arquivo DEX otimizado na cache Dalvik *(/data/data/Dalvik-cache)*. Essa otimização acontece quando o aplicativo é executado pela primeira vez após a instalação.

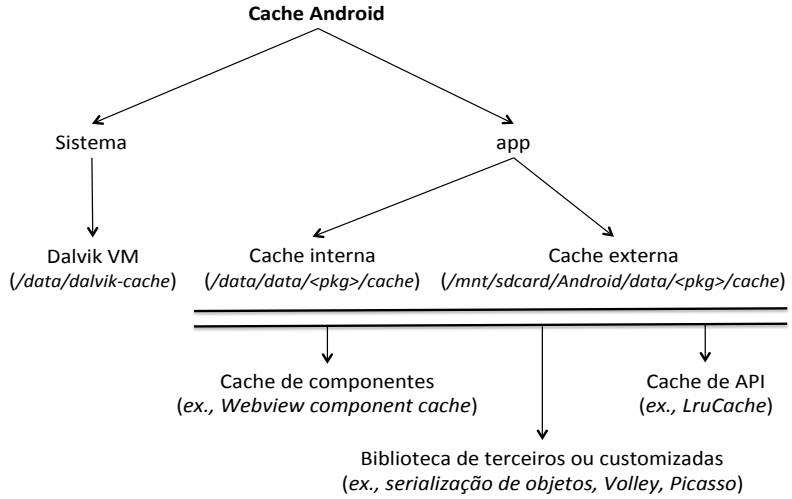

Figura 2.5: Organização da Cache no Android. Fonte: [21]

Já a Cache de **Aplicativos** depende do uso dado pelos desenvolvedores. Enquanto alguns desenvolvedores Android tratam especificamente de Cache em seus aplicativos, a maioria dos desenvolvedores nem sabem como seu aplicativo irá armazenar dados em Cache internamente, uma vez que usam APIs para esse fim. De forma geral, os desenvolvedores tem a opção de armazenar dados de componentes como *WebView* (para salvar dados de páginas Web), SQLite (para o gerenciamento de banco de dados), imagens, objetos serializáveis (como dados do youtube), entre outros.

2.2 Bibliotecas de Cache para Android

Embora na plataforma Android existam duas implementações disponíveis de Cache (**LRUCache** e **DiskLRUCache**), bibliotecas de terceiros tem sido criadas para melhorar/facilitar alguns aspectos da implementação ou atender algum requisito mais complexo. Tais bibliotecas são descritas e analisadas nesta seção.

2.2.1 LRUCache

A LRUCache, disponível como parte do Android SDK, é especializada na tarefa de gerenciar objetos em memória [26]. Armazena os dados (objetos) na memória e utiliza o algoritmo de substituição LRU (*Least Recently Used*) para manter objetos recentemente referenciados acessíveis e remover os menos recentemente utilizados antes que a Cache exceda o tamanho designado. Em outras palavras, cada vez que um dado já existente na Cache é acessado, ele é movido para o

início da fila. Caso não exista é adicionado no final da fila. Quando um dado que não existe precisa ser adicionado a um Cache completo, o dado no final da fila é despejado e o novo dado é posto em seu lugar.

Por exemplo, quando o usuário desliza para baixo uma lista de imagens no dispositivo, as imagens que aparecem são colocadas no topo da fila e as imagens que desaparecem são liberadas automaticamente. Isso contribui positivamente, pois as imagens em Cache aparecem quase que instantaneamente, diminuindo o consumo de energia.

Na LRUCache, o acesso a Cache é muito rápido (memória RAM), mas, por se tratar de um recurso limitado, sofre restrições quanto ao tamanho da Cache e o local de implantação [27]. Assim, sua implementação permite definir (como parâmetro) o tamanho máximo a ser usado em Cache. Além disso, possui métodos para verificar o tamanho da Cache e o tamanho de cada entrada.

2.2.2 DiskLRUCache

Diferente da LRUCache, a biblioteca DiskLRUCache é voltada para o gerenciamento de arquivos em disco (Memória interna do dispositivo ou SDCard). Para tanto, armazena os arquivos em um sistema de arquivos *journalling* [28], onde um arquivo de log ("journal") registra todas as modificações antes delas serem realmente salvas em disco. A DiskLRUCache limita o número de bytes que irá armazenar no sistema de arquivos. Assim, quando o número de bytes armazenados excede o limite, a Cache remove entradas até o limite ser satisfeito.

Na DiskLRUCache, o acesso a Cache é mais lento, porém não existem restrições quanto ao uso dos recursos [27]. Dentre as várias aplicações que utilizam a DiskLRUCache como sua API padrão, pode-se destacar o LinkedIn e o Skype. A DiskLRUCache pode ser encontrada no Diretório:

"/Android/libcore/luni/src/main/java/libcore/io/DiskLruCache".

2.2.3 Android Easy Cache

Android Easy Cache [27] é uma biblioteca que fornece a capacidade de implementar Cache tanto em memória RAM (usando LRUCache) quanto em disco (usando DiskLRUCache), permitindo-as trabalhar em conjunto.

A biblioteca permite operar com quatro opções de configurações para uso de RAM e Disco. Na configuração padrão (*Default Serializer*), a biblioteca trabalha salvando o conteúdo em formato JSON (*JavaScript Object Notation*). Na configuração *Custom Serializer*, o desenvolvedor pode usar formatos de terceiros. Além disso, na configuração *References*, a biblioteca permite que objetos armazenados em RAM sejam acessados na Cache através de referências, sem que

nenhuma serialização seja feita. Por fim, há a possibilidade do desenvolvedor habilitar ou desabilitar o uso em disco ou RAM (*Disable*) como parâmetro.

Os exemplos a seguir ilustram: (a) o uso da serialização padrão (*Default Serializer*) na RAM, mas sem uso de Disco; e (b) o uso de cache com referências (*References*) na RAM e um serializador padrão em Disco.

```
(a)
DualCache<AbstractVehicule> cache = new DualCacheBuilder<AbstractVehicule>(
      CACHE_NAME, TEST_APP_VERSION, AbstractVehicule.class)
          \\Default Serializer RAM
          .useDefaultSerializerInRam(RAM_MAX_SIZE)
          \\Disable
          .noDisk();

(b)
DualCache<AbstractVehicule> cache = new DualCacheBuilder<AbstractVehicule>(
      CACHE_NAME, TEST_APP_VERSION, AbstractVehicule.class)
          \\References
          .useReferenceInRam(RAM_MAX_SIZE, new SizeOfVehiculeForTesting())
          \\Default Serializer Disco
          .useDefaultSerializerInDisk(DISK_MAX_SIZE, true);
```

Vale ressaltar que: (i) por padrão, o criador da biblioteca recomenda uma Cache maior em disco do que em RAM; (ii) para o uso do *custom serializer* ou *references*, o cálculo para o tamanho do objetos deve ser feito pelo desenvolvedor, para ser capaz de executar corretamente a política LRU (*Least Recently Used*).

2.2.4 Simple DiskCache

A **Simple DiskCache** [29] é uma biblioteca que facilita o uso da DiskLRUCache. Segundo seu criador, a DiskLRUCache possui interfaces de baixo nível que são confusas e complexas para a maioria dos casos de uso. Por este motivo, a *Simple DiskCache* fornece interfaces mais genéricas (*Open*, *Put* e *Get*, por exemplo) para manipulação da Cache em disco.

Um ponto importante da biblioteca é que cada entrada de Cache contém um valor e metadados, representados como $Map < String, Serializable >$, e qualquer sequência pode ser usada como uma chave de Cache.

Vale ressaltar que o tamanho da cache é definido pelo desenvolvedor (método Open). Outros pontos sobre a *Simple DiskCache* é a falta de documentação e o uso como uma espécie de capa para a DiskLRUCache, onde sua contribuição foi em dar uma "nova roupagem", criando facilidades de implementação que a DiskLRUCache não possui.

2.2.5 HttpResponseCache

HttpResponseCache [30] é uma biblioteca que fornece Cache transparente e automática de requisições HTTP e HTTPS em disco, usando a biblioteca DiskL-

RUCache. A biblioteca utiliza o código *HttpResponseCache*, copiado do Projeto OpenSource do Android (AOSP) e modificado para trabalhar sob o padrão Java.

Por padrão, a biblioteca tenta fazer com que o tamanho da Cache não ultrapasse 10 MB, mas os autores afirmam que o melhor tamanho depende da aplicação e da frequência dos arquivos que estão sendo baixados.

Além disso, permite que aplicações usem Cache em armazenamento externo (opcional). Contudo, não há nenhum controle de acesso no diretório de armazenamento externo e, por isso, os autores recomendam que ela não deve ser usado para Caches que podem conter dados privados.

2.2.6 Qachee

Qachee [31] é uma biblioteca que fornece um sistema de Cache genérico em memória RAM para aplicações Android. Implementa o algoritmo LRU para gerência da memória através da biblioteca LRUCache.

O método construtor, na forma *private*, impede a instanciação por outras classes. Outro ponto é que a quantidade de memória a ser iniciada representa 1/8 da memória total disponível nos dispositivos, conforme descrito no código exemplo abaixo, extraído da Classe *QacheeManager.java*.

```
private QacheeManager() {
        final int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);
        // Usar 1/8 da memória disponível para este cache de memória.
        final int cacheSize = maxMemory / 8;

        qachee = new LruCache<>(cacheSize);
        expirationTime = ExpirationTime.ONE_MINUTE;
}
```

A biblioteca implementa uma política padrão de tempo de expiração (1 minuto) para as entradas na Cache, mas possui valores pré-definidos de 10 segundos, 30 segundos, 2 minutos, 10 minutos e 20 minutos.

2.2.7 Reservoir

Reservoir [32] é uma biblioteca simples para Android que permite serializar objetos Java e armazená-los em Cache, interna (RAM) e externa (disco), no formato JSON através da biblioteca GSON do Google. Isso permite a biblioteca acrescentar qualquer tipo de arquivo em Cache, desde que possa ser serializado via GSON.

Em linhas gerais, Reservoir é apenas um invólucro para a biblioteca DiskLRUCache. O tamanho da Cache é definido no momento da inicialização da biblioteca (via parâmetro), mas vale lembrar que é dependente diretamente da DiskLRUCache.

Na Reservoir, a limpeza da Cache pode ser total ou por entrada individual. Quando a Cache tem seu tamanho excedido, a política LRU é aplicada.

2.2.8 Android-BitmapCache

Android BitmapCache [33] é uma biblioteca especializada na criação de Cache para uso com objetos Bitmap, tanto em RAM (usando LRUCache) quanto em disco (DiskLRUCache).

Usa controle de políticas de reciclagem para limpar a Cache. Como padrão, o Bitmap é sempre reciclado quando os dados não estão sendo usados. O Bitmap é compactado com as configurações pré-estabelecidas pelo desenvolvedor. A codificação e decodificação do Bitmap ocorrem em processos distintos de acordo as demandas de gerenciamento da Cache.

Por padrão, a Cache para memória (RAM) vem ativada e para disco desativada. Na memória RAM o tamanho inicial é definido como 3MB (assim como outras configurações), conforme o código exemplo a seguir.

```
public final static class Builder {
        static final int MEGABYTE = 1024 * 1024;
        static final float DEFAULT_MEMORY_CACHE_HEAP_RATIO = 1f / 8f;
        static final float MAX_MEMORY_CACHE_HEAP_RATIO = 0.75f;
        static final int DEFAULT_DISK_CACHE_MAX_SIZE_MB = 10;
        static final int DEFAULT_MEM_CACHE_MAX_SIZE_MB = 3;
        ...
```

2.2.9 ObjectCache

A ObjectCache [34], derivada da Reservoir, é uma biblioteca que permite a criação de representações JSON de objetos (usando a biblioteca GSON), tanto em disco (usando DiskLRUCache) quanto em tempo de execução na memória RAM.

Como padrão, o tamanho máximo da Cache em Disco é 10MiB, podendo ser instanciado com outro valor. A limpeza de Cache pode ser feita de uma vez, especificando apenas o código *diskCache.clearCache*() (código exemplo a seguir).

```
String cachePath = context.getCacheDir().getPath();
File cacheFile = new File(cachePath + File.separator + BuildConfig.PACKAGE_NAME);

DiskCache diskCache = new DiskCache(cacheFile, BuildConfig.VERSION_CODE,
        1024 * 1024 * 10);
```

Opcionalmente, permite especificar o tempo em que as entradas expiram, variando de um segundo a um ano.

2.2.10 Kinvey

Kinvey [35] é uma biblioteca proprietária e paga que fornece meios para utilizar Cache, permitindo que a aplicação seja capaz de lidar com problemas de conectividade na rede e fornecendo recursos que deixam o aplicativo mais responsivo.

A biblioteca permite configurar o comportamento das políticas de Cache e, assim, determinar seu comportamento. A política *CachePolicy.NOCACHE* não utiliza qualquer Cache enquanto a *CachePolicy.CACHEONLY* permite recuperar dados a partir da Cache, mas não permite usar qualquer conexão de rede. A política *CachePolicy.CACHEFIRST* primeiro tenta recuperar dados da Cache e caso os dados foram armazenados em Cache, ele serão devolvidos e se os dados não existem em Cache, os dados serão recuperados do *backend* da Kinvey e a Cache será atualizada.

Como diferencial, a biblioteca implementa uma extensão de criptografia para Android, permitindo aplicar criptografia para **credenciais de usuário**, **arquivos** e **armazenamento SQLite offline** armazenados no disco. Estas classes podem ser usadas para substituir o comportamento padrão, usando criptografia sempre que algo é armazenado localmente.

2.2.11 Expirable DiskLRUCache

ExpirableDiskLruCache [36] é uma biblioteca associada a DiskLRUCache que permite a expiração por pares de chave/valor através de um tempo de despejo (*evictionTimeSpan*). Também permite acrescentar qualquer tipo de arquivo em Cache, desde que possa ser serializado via GSON. Como diferencial, permite criptografar dados em Cache usando a biblioteca Conceal [37] do Facebook, mas qualquer outra biblioteca personalizada pode ser empregada.

Caso o programador da aplicação queira usar uma outra biblioteca personalizada para codificação/decodificação (criptografia), a Classe *EncrypterDecrypter.java* deve ser alterada. Além disso, quando a Cache tem seu tamanho excedido, a política de LRU é aplicada.

2.2.12 Carbonite

Carbonite [38] é uma biblioteca para gerência de objetos persistentes, permitindo manter objetos Java puros (POJO) em memória enquanto são transparentemente armazenados em disco. Também permite especificar como e quanto tempo os dados serão mantidos.

A biblioteca permite definir, quando instanciada, o tamanho da Cache em Memória e em Disco, sendo este último armazenamento opcional.

A biblioteca aplica métodos síncronos e assíncronos, fazendo uso de políticas LRU com uso da LRUCache, DiskLRUCache e ainda serialização com uso da Kryo [39] (uma estrutura ágil e eficiente para serialização gráfica de objeto para Java).

2.2.13 Discussão

Após apresentar 10 (dez) bibliotecas de terceiros mais 2 (duas) de uso padrão do Android (LRUCache e DiskLRUCache), é fácil perceber uma falta de padronização em vários aspectos do uso das Caches. Primeiro, não existe um formato universal de Cache para aplicativos Android. Os desenvolvedores tem total liberdade para decidir qual o formato de Cache será usado ou é apropriado para suas aplicações. Segundo, uma mesma aplicação pode fazer uso de várias bibliotecas de Cache, dependendo das suas necessidades e dos componentes utilizados na mesma. Assim, podem ser usadas bibliotecas específicas para *caching* de imagem, *caching* de rede, entre outros.

Para ilustrar melhor, a Tabela 2.3 apresenta, de forma resumida, uma comparação entre as bibliotecas de Cache de terceiros, mostrando sua atividade (datas), os tipos de armazenamentos permitidos e características (vantagens e desvantagens).

Tabela 2.3: Comparação entre as Bibliotecas de Cache para Android

Biblioteca	**Data**		**Armazenamento**		**Características**
	Criação	**Atual.**	**Memória**	**Disco**	
Android Easy Cache	Mai/2014	Jun/2015	LRUCache	DiskLRUCache	Permite operar com quatro opções de configurações para uso de RAM e Disco, inclusive a possibilidade de habilitar ou desabilitar disco ou RAM
Simple DiskCache	Mar/2013	Abr/2014	Não permite	DiskLRUCache	Atua como uma casca para DiskLRUCache, facilitando, por exemplo, a definição do tamanho da Cache
HttpResponse Cache	Jan/2012	Out/2014	Não permite	DiskLRUCache	Fornece *Caching* transparente e automático de requisições HTTP e HTTPS. Permite limitar o tamanho da Cache
Qachee	Fev/2014	Jan/2015	LRUCache	Não permite	Permite especificar o tempo de expiração das entradas de Cache e limita o tamanho da Cache a 1/8 da memória do dispositivo
Reservoir	Dez/2013	Jun/2015	Não permite	DiskLRUCache	Permite especificar o tamanho da Cache, bem como a limpeza total ou por entrada individual da Cache
Android BitmapCache	Jul/2012	Nov/2013	LRUCache	DiskLRUCache	Permite especificar o tamanho da Cache e usa políticas de reciclagem para limpar a Cache
ObjectCache	Fev/2014	Jun/2015	Próprio	DiskLRUCache	Permite especificar o tempo de expiração das entradas de Cache e tem o tamanho máximo de 10Mb para Caches em disco
Kinvey	Não informada	Constante	LRUCache	SQLite 3	É proprietária e paga, mas aplica criptografia e tem suporte a 5Gb em disco
Expirable DiskLruCache	Mar/2015	Abr/2015	Não permite	DiskLRUCache	Permite a expiração de elementos individuais da Cache, bem como criptografar os dados em Cache
Carbonite	Jul/2013	Set/2014	Próprio (POJO)	DiskLRUCache	Permite especificar o tamanho da Cache em memória e disco

É importante ressaltar que as datas de criação e atualização correspondem as datas apresentadas pelos próprios criadores nos sites onde são disponibilizadas.

Ao analisar a Tabela 2.3 percebe-se que das 6 (seis) bibliotecas que fazem uso de Cache em memória, 4 (quatro) - Android Easy Cache, Qachee, Android Bitmap-Cache e Kinvey - utilizam a LRUCache como padrão. Somente a ObjectCache e Carbonite usam implementações próprias. Já na Cache em disco, 8 (oito) das 10 (dez) bibliotecas fazem uso da DiskLRUCache. A Qachee não armazena dados em disco e a Kinvey usa SQLite3 para isso.

Além disso, todas as bibliotecas listadas, com exceção da Kinvey, são livres e gratuitas. Embora não conste na tabela, as bibliotecas gratuitas estão liberadas sob duas licenças: MIT, para ObjectCache e Reservoir; e Apache (versão 2.0) para as outras. Por fim, no quesito segurança, apenas 2 (duas) bibliotecas (Kinvey e Expirable DiskLruCache) fazem uso de criptografia sobre os dados em Cache para aumentar a segurança e garantir a privacidade, confiabilidade e integridade dos dados.

2.3 Modelos Forense de Investigação

Com a proliferação do crime digital no mundo, inúmeros modelos e procedimentos de Investigação Forense Digital foram ou estão sendo desenvolvidos. Na prática, cada país tende a desenvolver seus próprios procedimentos, alguns com foco no aspecto da tecnologia outros focados na porção de análise dos dados da investigação [40, 41].

As subseções a seguir ilustram alguns modelos que podem ser utilizados como base para a construção de um novo modelo, apropriado e direcionado à investigação forense digital acerca dos dados contidos e/ou criados na cache das aplicações Android.

2.3.1 Modelo CFSAP - 1999

Em junho de 1999, McKemmish [42] enumera quatro grandes passos na investigação forense digital: (I) identificação de evidências digitais; (II) a preservação de evidências digitais; (III) a análise de evidências digitais; e (IV) a apresentação de evidências digitais. De acordo com [43], McKemmish, com a ajuda de outros autores, estendeu seu trabalho desenvolvendo o modelo CFSAP (*Computer Forensic - Secure, Analyze, Present*). O modelo CFSAP (Figura 2.6) fornece um quadro na qual os procedimentos ou processos forenses podem ser desenvolvidos individualmente.

O modelo combina os quatro elementos em três passos distintos. São eles:

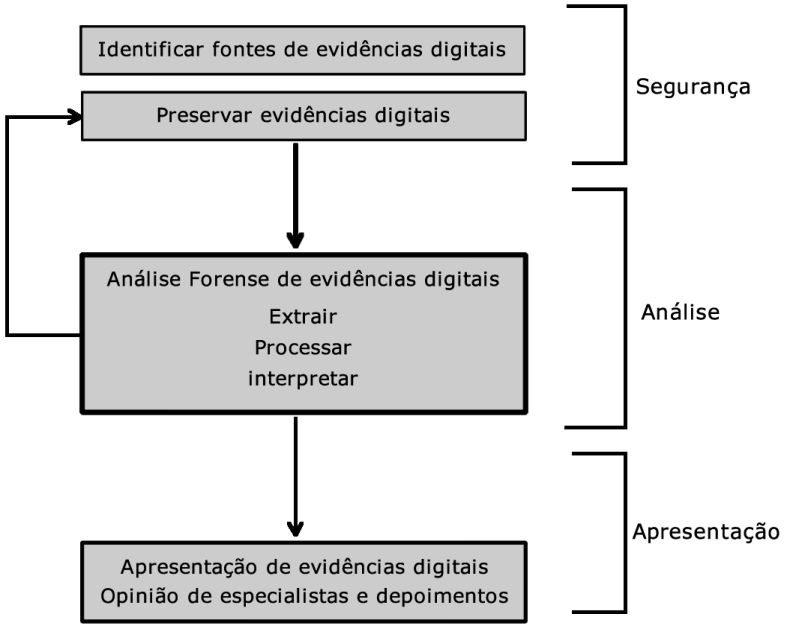

Figura 2.6: Modelo CFSAP. Fonte: [44]

1. **Obtenção de provas em potencial** - Abrange tanto a identificação de potenciais fontes de provas, bem como a preservação de dados que residem dentro de cada fonte. O foco principal desta fase é assegurar que toda a evidência disponível é identificada e capturada de modo que sua integridade e valor não seja diminuída [44]. Após a identificação é necessário garantir a originalidade, ou seja, a preservação, através de uma cópia exata dos dados, utilizando duas etapas: (a) Duplicação, usando técnicas forenses; (b) Autenticação da cópia, feita com uso de algoritmos OWHF *One-Way Hash Function*, que permite a comparação com o original, evitando uma futura contestação sobre modificações na cópia.

2. **Análise dos dados** - Abrange essencialmente três etapas. A primeira é a preparação dos dados, processo preparatório em que os dados capturados são disponibilizados para o processo de investigação, usando-se uma cópia autenticada do original. A segunda é o tratamento dos dados extraídos, onde é feita a busca de dados relevantes (envolvendo a digitalização de todos os dados preservados, em busca de informações que correspondem a um critério pré-determinado) e a extração de dados relevantes, o que envolve simplesmente o isolamento e a duplicação dos pontos relevantes dos dados (cópias). A terceira é a interpretação dos dados, realizada para estabelecer questões-chave, como a pertinência, contexto, propriedade e identidade, ca-

paz de expressar uma opinião que pode ser utilizado em processos judiciais.

3. **Apresentação dos resultados da análise** - É onde todos os dados relevantes obtidos, identificados, preservados e extraídos são apresentados como o resultado final da análise.

Para [43] esta modificação para criação do Modelo CFSAP foi considerada necessária porque os limites entre as duas fases, por vezes, não eram claras, relatando ainda outra diferença, mencionando o fluxograma usado para descrever o modelo CFSAP, pois garante aos usuários uma melhor compreensão dos procedimentos envolvidos em investigações forenses digitais, bem como um ciclo de retorno no gráfico que permite o movimento a partir da fase de **Análise** voltando para a fase de **Segurança** para garantir que as evidências digitais não serão esquecidas.

2.3.2 Modelo DFRWS - 2001

Durante a primeira oficina da Digital Forensics Research Workshop (DFRWS), em 2001, Palmer [45] apresentou um modelo que incorpora sete processos, cada um com seus próprios métodos e técnicas. A Figura 2.7 representa o processo de investigação forense digital.

Identificação	Preservação	Coleção	Exame	Análise	Apresentação	Decisão
Detecção de eventos/crime	Gerenciamento de Caso	Preservação	Preservação	Preservação	Documentação	
Deliberar Assinatura	Tecnologias de Imagem	Métodos Aprovados	Rastreabilidade	Rastreabilidade	Depoimento de um especialista	
Detecção de Perfil	Cadeia de Custódia	Software Aprovado	Técnicas de Validação	Estatística	Esclarecimento	
Detecção anômala	Sincronia de Tempo	Hardware Aprovado	Técnicas de Filtragem	Protocolos	Missão de estudo de impacto	
Reclamações		Autoridade Legal	Correspondência de padrões	Mineração de Dados	Contramedida Recomendada	
Monitoramento do sistema		Compressão sem perda	Descoberta de dados ocultos	Link	Interpretação Estatística	
Análise de Auditoria		Amostragem	Extração de dados ocultos	Espacial		
Etc.		Redução de Dados		Redução de Dados		
		Técnicas de Recuperação				

Figura 2.7: Modelo DFRWS. Fonte: [45]

O autor descreve que os itens ilustrados na Figura 2.7 estabelecem um processo linear, desde a identificação à decisão, que são as principais categorias ou

classes, que aparece para ser usado em análise forense digital. Abaixo de cada coluna estão as técnicas e métodos. Esses itens em cinza estão sujeitos a menos confusão, embora ainda haja alguma discussão sobre o uso do termo coleção e preservação e se é realmente uma subcategoria do outro. [45].

Para Palmer [45], tais atividades são de natureza investigativa, cujo intuito é empregar essas ferramentas e métodos no processo de investigação. Caso tais processos sejam categorizados de forma correta, estes permitem que os profissionais possam visualizar o que é necessário para ser acrescentado, bem como serve para pesquisadores em busca de falhas em tecnologias.

De acordo com [43], em seu artigo o autor menciona sobre o modelo DFRWS, citando seis processos ao invés de sete como mostrado na Figura 2.7, onde podemos verificar que a fase de decisão é uma categoria, então na verdade existem sete processos (categorias) principais, que elencam técnicas e métodos, que são descritas a seguir:

- **Identificação**: Evento/detecção de crime, resolução de assinatura, detecção de perfil, detecção de anomalias, resolução de litígios, monitoramento do sistema e análise de auditoria.
- **Preservação**: O gerenciamento de casos, a imagem latente, a cadeia de custódia, a sincronização de tempo.
- **Coleção**: Preservação, métodos aprovados, softwares e hardwares aprovados, autoridade legal, compressão sem perda, amostragem, redução de dados e recuperação.
- **Exame**: Preservação, rastreabilidade, validação, filtragem, correspondência de padrão, a descoberta de dados ocultos, extração de dados escondidos.
- **Análise**: Preservação, rastreabilidade, métodos estatísticos, protocolos, mineração de dados, cronograma, link e espacial.
- **Apresentação**: Documentação, testemunhos de peritos, esclarecimento, impacto de missão, recomendação de contra medidas e interpretação estatística.
- **Decisão**: Como visto na Figura 2.7 , este item não contem técnicas ou métodos elencados.

2.3.3 Modelo ADFM - 2002

Reith *et al.* [46] propuseram um modelo de processamento chamado *Abstract Digital Forensics Model* (ADFM), uma extensão do modelo DFRWS, que além

de destacar as características únicas do modelo DFRWS, apresenta ideias forenses tradicionais, em especial, o uso do protocolo do FBI[2] de investigação de cenas de crime.

De acordo com o modelo ADFM, uma investigação forense digital tem nove etapas principais: (1) **Identificação** - determinação do tipo de incidente; (2) **Preparação** - organização das ferramentas e técnicas necessárias, bem como mandados de busca; (3) **Estratégia de aproximação** - construção de uma abordagem para maximizar o recolhimento de provas e minimizar o impacto da vítima; (4) **Preservação** - proteção e manutenção do estado atual das evidências; (5) **Coleção** - gravação da cena física do crime e produção uma imagem duplicada de evidências digitais através de procedimentos qualificados; (6) **Exame** - realização de pesquisa avançada de provas relevantes do incidente; (7) **Análise** - fornecimento de uma interpretação das provas para construção da hipótese investigativa e para oferecer conclusões baseadas em evidências; (8) **Apresentação** - fornecimento de explicações acerca das conclusões; e (9) **Retorno da evidência** - devolução dos ativos físicos e digitais aos seus proprietários.

2.3.4 Modelo de Ciardhuáin - 2004

Ciardhuáin [47] destaca que os modelos existentes não abrangem todos os aspectos da cyber investigação, concentrando-se apenas no processamento de evidências digitais, que embora valioso, não é suficiente para descrever completamente o processo de investigação de forma que possa colaborar com o desenvolvimento de novas ferramentas e técnicas de investigação.

Segundo o autor, um modelo mais abrangente possível pode fornecer um quadro de referência comum para discussão e para o desenvolvimento de terminologias, podendo vir a apoiar o desenvolvimento de ferramentas, técnicas, formação e certificação/credibilidade dos investigadores e ferramentas. Pode também fornecer um sistema unificado com uma estrutura para estudo de caso ou lições aprendidas, bem como materiais a serem compartilhados entre os pesquisadores para criação de padrões a partir da avaliação de conformidade das melhores práticas de investigação.

O modelo possui 13 processos: Consciência; Autorização; Planejamento; Notificação; Procurar e identificar a evidência; Coleta de provas; O transporte de provas; Armazenamento de provas; O exame de provas; Hipótese; Apresentação da hipótese; A prova/Defesa da hipótese; Divulgação da informação.

[2]http://www.fbi.gov/about-us/lab/forensic-science-communications/fsc/april2000/twgcsi.pdf

2.3.5 Modelo Forensic Process - 2006

Kent *et al.* [48] propuseram um processo de investigação dividido em 4 fases: **Coleta**, onde os dados relativos a um evento específico é identificado, marcado, gravado e recolhido, e a sua integridade é preservada; **Exame**, onde ferramentas forenses e técnicas apropriadas para os tipos de dados que foram coletados são executadas para identificar e extrair as informações relevantes a partir dos dados coletados e ao mesmo tempo proteger sua integridade; **Análise**, que envolve a análise dos resultados do exame para obtenção de informação útil que aborda as questões que foram o impulso para realizar a coleta e exame; e **Relatórios**, que descrevem as ações realizadas, determinando o que outras ações precisam ser feitas e recomendando aprimoramento de políticas, diretrizes, procedimentos, ferramentas e outros aspectos do processo forense.

De acordo com o autor, estes processos transformam as mídias (dados coletados) em evidência seguindo 3 passos. No primeiro, os dados coletados são analisados, ou seja, extraídos das mídias e transformados em um formato que podem ser processados por ferramentas forenses. No segundo, através da análise, os dados são transformados em informação. Por fim, ocorre a transformação da informação em ação usando a informação produzida pela análise uma ou mais maneiras durante a fase de relatórios.

2.3.6 Discussão

Antes de iniciar a discussão sobre os modelos, é importante ressaltar que existem, no mínimo, algumas dezenas de modelos de análise de forense digital e que por isso este trabalho optou por abordar somente as que poderiam de alguma forma servir de base para a construção de um novo modelo direcionado à investigação forense digital de possíveis vazamentos de dados a partir da cache das aplicações Android.

Partindo deste pressuposto, percebe-se que os modelos citados possuem elementos que são comuns entre eles, recebendo apenas uma nova nomenclatura, talvez como uma forma encontrada para melhorar o modelo ou separar subprocessos que não eram contemplados. A Tabela 2.4 ilustra os modelos apresentados, tomando por base o modelo de Ciardhuáin [47] que possui a maior quantidade de elementos (13 processos), sendo o mais abrangente em termos de uma cyber investigação, englobando assim, as demais fases dos modelos apresentados, ou seja, Ciardhuáin apenas destrinchou ainda mais as fases dos modelos já existentes, como pode ser observado na Tabela 2.4.

O modelo de Ciardhuáin [47] enfatiza que o estudo de uma cena de crime deve ser sistemático e metódico e está voltado principalmente para investigações

Tabela 2.4: Elementos de comparação entre os modelos

<table>
<tr><th rowspan="2">Fases de Ciardhuáin</th><th colspan="4">Modelos</th></tr>
<tr><th>CFSAP</th><th>DFRWS</th><th>ADFM</th><th>Forensic Process</th></tr>
<tr><td>Consciência</td><td rowspan="8">Identificação</td><td rowspan="4">Identificação</td><td rowspan="4">Identificação, Preparação e Estratégia de aproximação</td><td rowspan="8">Coleta</td></tr>
<tr><td>Autorização</td></tr>
<tr><td>Planejamento</td></tr>
<tr><td>Notificação</td></tr>
<tr><td>Procurar e identificar a evidência</td><td>Coleção</td><td>Coleção</td></tr>
<tr><td>Coleta de provas</td><td rowspan="5">Preservação, Exame e Análise</td><td rowspan="3">Preservação</td></tr>
<tr><td>Transporte de provas</td></tr>
<tr><td>Armazenamento de provas</td><td rowspan="2">Preservação e Análise</td></tr>
<tr><td>Exame de provas</td><td>Exame</td><td>Exame</td></tr>
<tr><td>Hipótese</td><td rowspan="4">Apresentação</td><td rowspan="3">Apresentação e Decisão</td><td rowspan="3">Análise</td><td rowspan="3">Análise</td></tr>
<tr><td>Apresentação da hipótese</td></tr>
<tr><td>Prova/Defesa da hipótese</td></tr>
<tr><td>Divulgação da informação</td><td>Apresentação e Retorno da evidência</td><td>Relatórios</td></tr>
</table>

usando evidências físicas, mas que também reflete em muitos aspectos de um exame forense eletrônico. A sua limitação é que se refere apenas a parte forense de uma investigação e pontos como política externa, regulamentação e legislação são questionáveis por não estarem vinculadas a políticas de controle das informações.

Os processos de identificação, preservação, análise e apresentação do modelo CFSAP, desenvolvido por McKemmish [42], são elementos contidos e ainda "comuns" nos demais modelos. Dentre as metodologias descritas, é notório que as fases de recolhimento e análise são passos integrais no processo forense digital. No caso de um dispositivo móvel, elas baseiam-se na análise da imagem (cópia do dispositivo) recolhida. De acordo com [49], na prática, essas fases correspondem a uma imagem inicial Android criada para extrair partições, incluindo Cache e dados forense, sem interferir com os dados do dispositivo.

Após avaliar várias referências e material bibliográfico, constatou-se não existe nenhum processo ou técnica específica com enfoque diretamente a análise da Cache dos aplicativos da plataforma Android. Tal fato é um dos motivadores para realização deste trabalho.

Capítulo 3

Trabalhos Relacionados

Este Capítulo realiza uma breve análise dos trabalhos relacionados sobre Cache em Android. Vale a pena relatar que, após uma revisão sistemática, poucos trabalhos remetem ao pretendido, que por sua vez, não trabalham com a análise de dados de Cache para a plataforma Android. Desta forma, a ideia posterior foi revisar na literatura trabalhos que relatem algum tipo de análise voltado para dispositivos móveis com foco em detecção de vazamento de informações na plataforma Android.

Conforme a revisão realizada, encontrou-se trabalhos que empregam mecanismos com o propósito de detectar ou coibir vazamento de informações, onde é descrito alguns de forma resumida e posteriormente é apresentado os que de alguma forma trabalham com Cache de aplicações na plataforma Android.

Sendo assim os trabalhos apresentados serão divididos nos que empregam análise de marcação *taint analysis* para detectar possíveis vazamentos no código dos aplicativos e posteriormente os trabalhos baseados no Cache das aplicações.

3.1 Análise de Marcação (*Taint Analysis*)

Segundo a OWASP [50], *taint analysis* tenta identificar as variáveis que foram "contaminadas" a partir de uma entrada definida (controlável) pelo usuário e assim traçar possíveis funções vulneráveis, também conhecida como "sumidouro" (*sink*). Se a variável contaminada é passada para um sumidouro sem primeiro ser higienizada, ela é marcada como uma vulnerabilidade.

A solução Intitulada *TrustDroid* [51] realiza uma análise semântica estática no *bytecode* do Android para monitorar entradas que manipulem dados sensíveis. A principal contribuição do TrustDroid é que não há necessidade de qualquer alteração no sistema operacional, bem como em pacotes (apk). Por outro lado, ele depende de intervenção do usuário para seu funcionamento. Além disso,

apresenta um consumo de bateria excessivo quando utilizado em tempo real, bem como a falta de capacidade de analisar aplicativos que foram construídos com bibliotecas não nativas.

No trabalho de Yang e Yang [52] é proposta uma abordagem (*LeakMiner*) capaz de detectar vazamento de informações confidenciais no Android, mas verificando os aplicativos ainda nos sites de lojas (por exemplo, Google Play Store) e evitando que aplicativos maliciosos sejam distribuídos para os usuários. A principal contribuição do LeakMiner é proporcionar análise antes da instalação do aplicativo no dispositivo do usuário, ainda na loja, eliminando problemas de desempenho e consumo de recursos (bateria, principalmente) que existem em outras abordagens. Já a principal limitação é a alta taxa de falsos positivos. Nos testes foram usadas 305 aplicativos que vazam informações. O LeakMiner acertou corretamente 145 delas, mas errou 160, o que dá uma taxa de erro de 52%.

O *Appintent* de Yang *et al.* [53] é uma ferramenta capaz de definir se a transmissão de dados sensíveis do usuário em um aplicativo Android foi consentida ou não. A principal contribuição foi o desenvolvimento de uma plataforma de análise dinâmica para executar uma aplicação Android com entradas de dados e eventos, onde a sequência de manipulações podem ser realizadas e visualizadas na interface do usuário, sendo o sistema capaz de emular todo o processo para transmissão de dados sensíveis e assim detectar o vazamento. A principal limitação é a dependência da decisão de um testador humano.

Zhang e Yin[54] propuseram o desenvolvimento de um mecanismo em tempo de execução eficiente para análise do fluxo de informações e políticas de segurança em aplicativos Android, sem qualquer modificação no *firmware* e com baixa sobrecarga no tempo de execução. Resumidamente o método emprega uma *engine* (BRIFT) que reescreve qualquer aplicativo Android para inserir instruções no *bytecode* capazes de monitorar fluxo de informações confidenciais.

A principal contribuição é a forma inovadora da solução, uma abordagem de reescrita do *bytecode* sem a necessidade de qualquer alteração no *firmware* do dispositivo, bem como impacto mínimo no desempenho. Já a principal limitação é que ela não consegue lidar com componentes nativos e chamadas *Java* de forma reflexiva de maneira geral. Além disso, a reescrita de código pode levar bastante tempo.

O *TaintDroid* [55] é uma ferramenta de análise em tempo real que aproveita o ambiente de execução virtualizado do Android para monitorar dados sensíveis utilizados em aplicações, utilizando marcações taint de forma dinâmica. De modo geral, o TaintDroid identifica fontes sensíveis de privacidade como IMEI, localização, entre outros, e as marca. Mesmo sendo a primeira ferramenta a usar rastreamento *taint* dinâmico para detectar vazamento de dados em aplicativos do Android, ela possui limitações. Primeiro, seus métodos incorrem certa

sobrecarga de desempenho no sistema. Segundo, TaintDroid não detecta vazamentos em aplicativos que usam o código de bibliotecas não nativos. Terceiro, provoca falsos positivos significativos quando as informações rastreadas contém identificadores de configuração.

Os autores de *FlowDroid* [56] propuseram uma ferramenta de análise taint estática para aplicações Android. A ideia por trás do FlowDroid é, após descompactar um pacote, procurar o ciclo de vida dos métodos de chamada e pedidos de retorno (*callback*), bem como chamadas para fontes (*sources*) e sumidouros (*sinks*). Esta análise é realizada em diversos arquivos Android, incluindo arquivos *XML* de layout, arquivos *dex* contendo o código executável e o arquivo de manifesto.

Os autores afirmam que o FlowDroid é a primeira ferramenta de análise taint estática que utiliza contextos precisos, fluxo, campo, objetos sensíveis e o ciclo de vida completo para Android. Por outro lado, sua principal limitação é não conseguir lidar com as chamadas reflexivas (apenas se forem cadeia de strings) e seu método pode ser contornado se alguma chamada de retorno desconhecida for utilizada.

No *AndroidLeaks* [18] os autores projetaram uma estrutura de análise estática para identificar possíveis vazamentos de informações pessoais em aplicativos Android em grande escala. A ideia é transformar os arquivos *.dex* em arquivo *.java*, criando um conjunto mapeável entre métodos da API Android e as permissões contidas no arquivo de manifesto *.xml* que acompanha o pacote (apk). Após este mapeamento é realizada a análise do fluxo de dados, detectando que tipos de vazamentos estão ocorrendo e, por fim, gerando um relatório com tais dados.

A contribuição do trabalho é que a ferramenta pode encontrar de forma automática possíveis vazamentos em uma escala maciça, resultando em uma análise massiva de 800 aplicativos por hora, processando coletivamente mais de 531.249 classes java. A limitação da ferramenta está relacionada aos entraves inerentes à análise estática, pois seus resultados erram no lado de falsos positivos ao invés de falsos negativos. Os próprios autores reconhecem que a combinação com a análise dinâmica seria a em tese a solução deste problema. Outro ponto é que *AndroidLeaks* não consegue analisar determinados fluxos de dados, como por exemplo objetos de intenção (*Intents Object*).

McClurg et al. [57] descrevem um protótipo em *Java*, baseado em desktop, capaz de instrumentar uma aplicação com a funcionalidade de propagação taint, fazendo com que todas as impurezas possam ser definidas/propagadas com o aplicativo que é executado na máquina virtual (Dalvik VM) do dispositivo. Primeiramente o pacote (apk) é escaneado para que os arquivos componentes da Dalvik VM sejam desmontados. Após ser desmontado, o sistema analisa o código de montagem e insere as atribuições apropriadas de propagação/coloração

ou chamam a biblioteca de propagação *taint* correspondente para cada instrução. Assim, um novo pacote é criado contendo o código assembly instrumentado em conjunto com a biblioteca de propagação *taint*. Após ser assinado usando um par de chaves pública/privada especificado pelo usuário, o aplicativo pode ser instalado no dispositivo do usuário.

A principal contribuição deste trabalho, segundo os autores, é em relação a abordagem rápida, pois a análise é realizada enquanto o programa está em execução. A limitação do trabalho está em não poder modificar o tempo de execução do Android, comum em outros sistemas, além de estarem obrigados a tratar chamadas do sistema Android como uma caixa preta, assumindo sempre propagar valores *taint*, significando assim, uma sobre-aproximação semântica do programa real que podem levar a falsos positivos.

A ferramenta IccTA [58] é descrita como um analisador *taint* estático para detectar vazamentos de privacidade na comunicação inter-componente (ICC) em aplicações Android, suportando qualquer análise de fluxo. Transforma o *bytecode .dex* em *Jimple* do *Framework Soot* [59] e depois extrai os links *ICC* armazenando-o em banco de dados. Em seguida modifica a representação *Jimple* para ligar diretamente os componentes para permitir a análise de fluxo de dados entre eles e usa uma versão modificada do FlowDroid [56] para construir um gráfico de controle de fluxo completo da aplicação.

As principais contribuições do IccTA incluem o desenvolvimento de um código de fonte aberto de análise *taint* para comunicação inter-componente (ICC), bem como uma nova metodologia para resolver o problema de ICC diretamente em nível de código. Quanto as limitações, o IccTA não resolve chamadas reflexivas (somente se os seus argumentos são string constantes), assim como não aceita *multi-threading*. Também apresenta a mesma limitação do FlowDroid para chamadas nativas. IccTA não lida com alguns métodos ICC raramente utilizados, tais como *startActivities* e *sendOrderedBroadcastAsUser*; não pode resolver operações complicadas de cadeia (por exemplo, usando *StringBuilder*) e a análise de strings é dentro de um único método que pode causar alarmes falsos.

O *Intentfuzzer* [60] propõe uma abordagem capaz de detectar vazamento de informações aplicando técnicas *fuzzing*[1], onde são gerados objetos de intenção (*Intent objects*) apropriados, de forma dinâmica, e enviados para os componentes com intuito de verificar se algum vazamento de fato está ocorrendo.

A principal contribuição do trabalho é que enquanto a análise estática (vista nos trabalhos de análise *taint*) só vê as possíveis ligações de chamadas entre chamadas de função, a técnica "*fuzzing dinâmica*" pode detectar vazamentos de

[1]Técnica de teste automatizada que abrange inúmeros casos de contorno usando dados inválidos (a partir de arquivos, protocolos de rede, chamadas de API e outros alvos) como entradas de uma aplicação para melhor garantir a ausência de vulnerabilidades exploráveis. [61]

permissão que realmente acontecem, que podem ser gravados e utilizados para reconstruir todas as cenas. A limitação do trabalho está relacionada ao fato de não poder procurar por vazamentos de capacidade em serviços vinculados, bem como não consegue lidar com *Broadcast Receiver* que estejam matriculados em tempo de execução. Outro ponto é que *IntentFuzzer* só detecta vazamentos de permissão quando um componente executa uma operação privilegiada imediatamente depois de receber uma *Intent*, produzindo assim resultados falso negativo.

3.1.1 Discussão

Para melhor e aprofundar a discussão sobre análise de marcação (análise *taint*), a Tabela 3.1 sumariza os trabalhos apresentados, indicando o tipo de análise empregada (estática ou dinâmica), se a detecção de vazamento é em tempo real ou off-line, e a fonte da análise (bytecode diretamente, bytecode convertido em código *java*, manifesto e Cache).

Tabela 3.1: Sumarização dos trabalhos apresentados sobre *Taint Analysis*

Autor	**Tipo de Análise**		**Deteção**		**Fonte de Análise**			
	Estática	**Dinâmica**	**Tempo Real**	**Off-line**	**Bytecode**	**Java**	**Arquivo Manifesto**	**Cache**
TrustDroid [51]	✓		✓	✓	✓		✓	
LeakMiner [52]	✓			✓		✓	✓	
AppIntent [53]	✓		✓		✓		✓	
Efficient [54]	✓		✓			✓	✓	
TaintDroid [55]		✓	✓		✓			
FlowDroid [56]	✓			✓		✓	✓	
AndroidLeaks [62]	✓			✓		✓	✓	
APLD [57]		✓	✓			✓	✓	
IccTA [58]	✓			✓		✓	✓	
Intentfuzzer [60]		✓	✓		✓		✓	

Pode-se observar que os trabalhos relacionados possuem características como: (i) tipo de análise (estática ou dinâmica); (ii) tipo de detecção (tempo real ou offline) e; a (iii) fonte de análise (bytecode, java, arquivo de manifesto e Cache). A análise realizada pela maioria (seja ela estática ou dinâmica) é em função do código da aplicação (bytecode), realizando testes semelhantes aos de caixa branca. Percebe-se também que o arquivo de manifesto é bastante relevante.

No que diz respeito a Cache, fica claro que nenhum dos trabalhos que usam análise de marcação aborda algum tipo de análise de dados de Cache das aplicações dos dispositivos móveis, pois o intuito é avaliar apenas as funções de retorno (*callback*) dispostas no código da aplicação.

3.2 Cache das Aplicações

Como mencionado anteriormente, existem diversos aplicativos disponíveis na Google Play Store (Clean Master, App Cache Cleaner, CCleaner, Cache Cleaner, entre outros) que trabalham fazendo limpeza de Cache, oferecendo inclusive outros serviços como antivírus, proteção de aplicativos (AppLock), gerenciador de app e muito mais. Todos desenvolvidos para fins comerciais e nenhum com preocupações de segurança.

O primeiro trabalho a se preocupar com a segurança do usuário e relacionar isso a Cache foi o de Amini et al. [63]. Os autores propuseram a criação de um mecanismo (**Caché**) capaz de realizar uma pré-busca de dados e deixá-los disponíveis em Cache para uso posterior, não permitindo assim a liberação de informações sobre a localização do usuário. A ideia parte do princípio que o desenvolvedor de uma aplicação registra o uso do Caché, declarando suas necessidades de conteúdo e o formato de solicitação. Após instalar a aplicação (app) que faz uso do Caché, o usuário especifica a região para a qual o conteúdo deve ser baixado, especificando um endereço como casa, trabalho ou um código postal. A app usando Caché então faz o download do conteúdo para a região especificada. Desta forma, o conteúdo passa ser utilizado de modo offline, sem necessidade de conexão de dados. Segundo os autores, a arquitetura do Caché é semelhante a um proxy Internet não transparente.

A Figura 3.1 exemplifica o funcionamento da Caché.

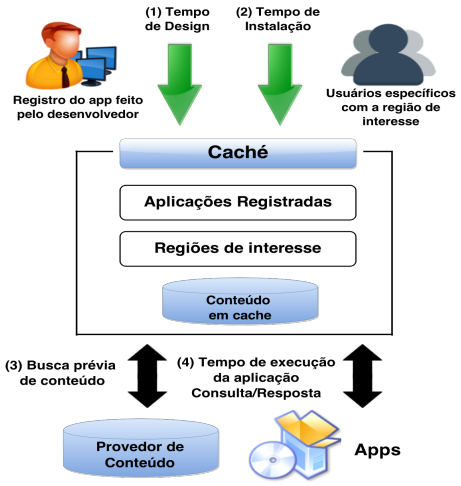

Figura 3.1: Funcionamento do Caché. (1) Uma vez que o desenvolvedor tenha registrado a aplicação (2) e o usuário ter especificado as regiões para quais conteúdos devem ser armazenados em cache, (3) o Caché faz o pedido e (4) armazena o conteúdo para uso futuro pela aplicação. Fonte: [63].

A principal contribuição do trabalho está na viabilidade de usar Cache para manter a privacidade dos usuários, através do uso da pré-busca que permite utilizar conteúdos com capacidade de localização, garantindo a privacidade do usuário, sem contar com os benefícios relacionados ao download de dados. Outro ponto positivo é a elaboração de uma taxonomia de tipos de dados baseados em localização e uma discussão sobre vantagens e desvantagens em relação a atualização de dados, armazenamento e requisitos de banda larga.

A limitação do trabalho consiste na não existência de garantias a privacidade dos dados do usuário se a aplicação somente usar dados online, como é o caso das aplicações de check-in com localização, como Foursquare[2]. Outro ponto a ser considerado está relacionado ao custo de baixar dados para cache que não satisfaçam a necessidade do usuário (um restaurante que não existe mais, uma rota de ônibus, etc.).

Finley e Xiaojiang [1] propuseram uma forma dinâmica de realizar a limpeza da Cache em plataforma Android. Para tanto, desenvolveram uma ferramenta, denominada *Dynamic Cache Cleaning* ou **DCC**, cuja função é remover arquivos em cache de forma mais agressiva, sem a necessidade de interação com o usuário. Assim, o DCC libera memória interna através de um serviço em *background* e transparente, resultando no melhor desempenho do dispositivo. Além disso, oferece maior segurança para o dispositivo e o usuário, removendo arquivos potencialmente sensíveis de forma rápida e eficaz.

Os autores afirmam que embora existam várias aplicações para limpeza de cache em Android, elas dependem que o usuário defina intervalos específicos para remover os dados em cache. Por outro lado, DCC é executada com base em uma fórmula para calcular o momento mais adequado da limpeza. A fórmula considera apenas dois fatores: o tamanho da cache e ociosidade do dispositivo. Desta forma, quando o tempo de inatividade (ociosidade) do dispositivo e o tamanho da cache de uma aplicação atinge um certo limite, o cache do aplicativo é completamente limpo.

A Figura 3.2 representa o fluxograma do Dynamic Cache Cleaning.

Sempre que uma aplicação possuir dados de Cache, o DCC registra um evento e fica monitorando por duas ações específicas. A primeira é o aplicativo poder ser reaberto, o que obviamente sinaliza que a aplicação está ativa e impede a limpeza de seu cache. A segunda é a data de expiração dos dados em cache dos aplicativos ultrapassar o limite definido e, assim, a DCC entrará em ação para limpar o cache do aplicativo.

A principal contribuição do trabalho consiste na forma dinâmica em limpar arquivos obsoletos em cache, evitando o consumo excessivo de espaços de memória

[2]http://pt.foursquare.com

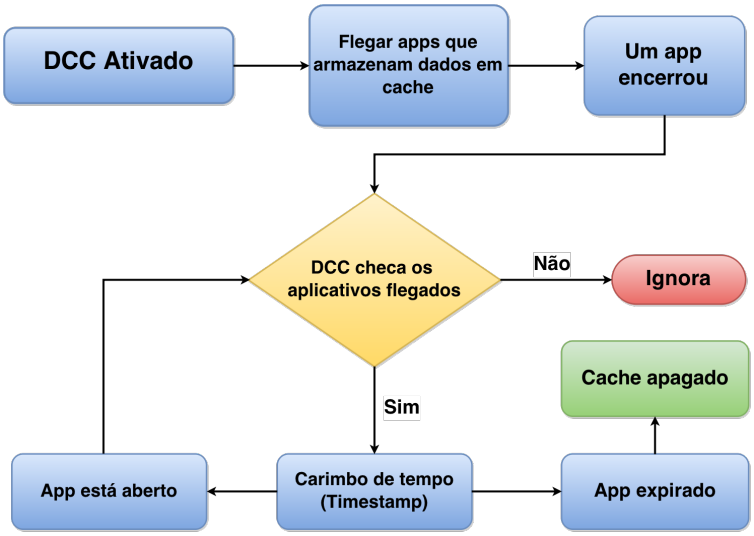

Figura 3.2: Fluxograma do DCC. Fonte: [1]

e melhorando a segurança dos usuários, uma vez que as práticas de programação pobres podem levar a cache a conter informações confidenciais. Por outro lado, a limitação da ferramenta realiza apenas a limpeza agressiva e rápida da Cache sem analisar se os dados existentes são sensíveis ou não.

Os autores em [64] realizaram uma pesquisa abrangendo mais de mil aplicativos Android, visando identificar como é a atuação do *Web Caching* nestas aplicações. Com base nas observações das aplicações, projetaram o **CacheKeeper**, um serviço transparente de *Web Caching* para aplicativos móveis.

A abordagem consiste em reduzir o trabalho para os desenvolvedores, fornecendo o CacheKeeper como uma camada de serviço no Kernel do Linux, onde os desenvolvedores não precisam instalar qualquer biblioteca adicional ou incorporar qualquer API, conforme ilustrado na Figura 3.3.

Os componentes da arquitetura do CacheKeeper são os seguintes:

- **HTTP Transaction Handler - HTTP-TH** - trata as solicitações HTTP oriundas dos aplicativos e as respostas HTTP recebidas da rede. Assim, o HTTP-TH consulta o componente Cache Manager para verificar se alguma resposta encontra-se em cache e caso não esteja repassa as respostas recebidas para o armazenamento em cache.

- **Cache Manager** - aceita e processa consultas de resposta em Cache a partir do HTTP-TH. Ele aceita respostas recém-vindas do HTTP-TH, armazenando-as num cache físico adequado e executando a substituição de cache se necessário.

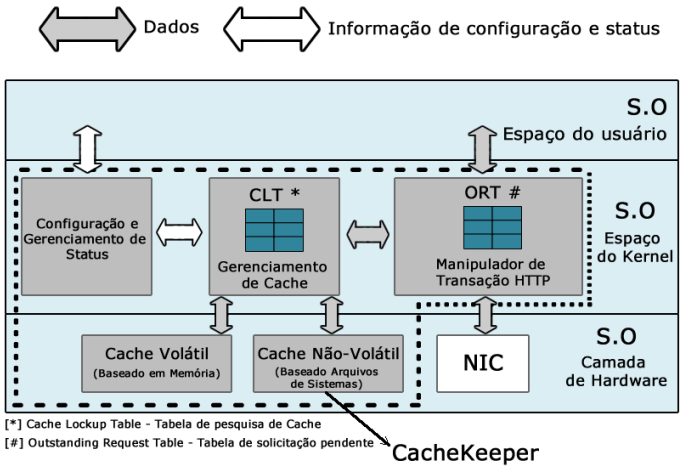

Figura 3.3: Arquitetura CacheKeeper. Fonte: Adaptado de [64].

- **Gerenciador de Configuração e Status** - fornece interfaces para programas configurarem o CacheKeeper para consultar o status de funcionamento (para fins de depuração).
- **Caches físicas** - o CacheKeeper suporta dois tipos de mídia de armazenamento em Cache: Cache volátil residente na memória do dispositivo e Cache não volátil residente em sistema de arquivos do dispositivo. A Cache volátil é a adequada para pesquisa eficiente e a Cache não volátil serve para garantir o conteúdo Cache persistente após reinicializações.

A principal vantagem da solução é permitir que todos os aplicativos que geram tráfego Web possam fazer uso, permitindo um *cross-cache* de aplicações e permitindo que um aplicativo tire vantagens de dados em Cache de outro aplicativo.

Obviamente, esse uso comunitário da Cache é um ponto negativo no aspecto de segurança. Embora o CacheKeeper forneça meios para permitir que os aplicativos especifiquem quais objetos HTTP podem ser armazenados no serviço de Cache, manteria o controle da privacidade dos usuários, tal mecanismo não é apresentado, muitos menos seus resultados em possíveis cenários de ataque.

Os autores do artigo "A Case for Application-Managed Cache for Browse" [65] afirmam que os aplicativos Web não têm controle sobre o armazenamento em Cache, levando à utilização ineficaz de cache . Partindo dessa constatação, os autores desenvolveram um Cache hierarquizado no lado do cliente, gerenciado por um aplicativo, chamado de "**HCache**". O HCache aproveita as opções de armazenamento do HTML5 como *back-ends* e permite que os desenvolvedores

de aplicativos Web possam controlar o comportamento de Cache Transparente e o uso dessas opções de armazenamento de forma inteligente. Os resultados do HCache mostram um aumento no desempenho em até 60% de melhoria das páginas Web.

O HCache mapeia chaves para objetos arbitrários. Cada objeto também tem metadados, que contém informações relacionadas à validade, validação e despejo. A Figura 3.4 ilustra uma visão geral do HCache.

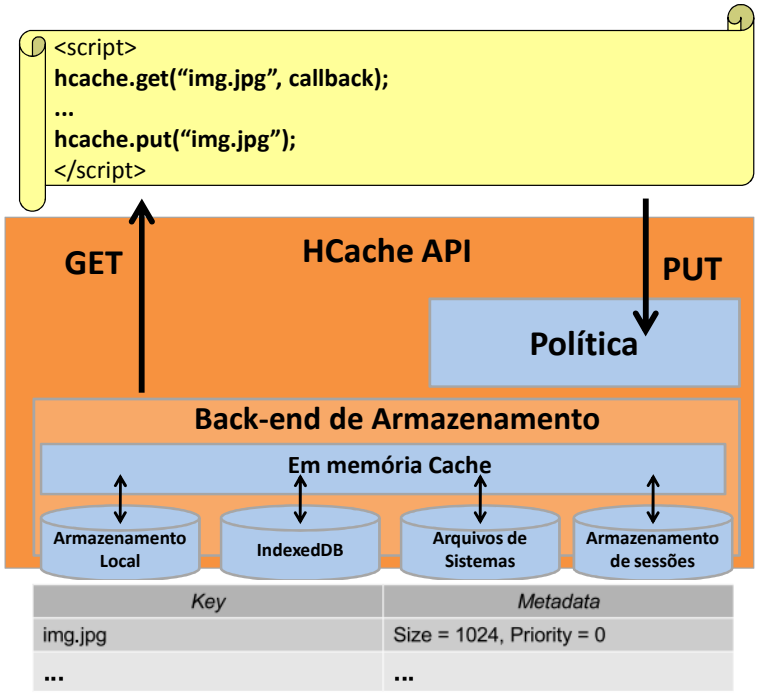

Figura 3.4: Visão Geral HCache. O HCache encapsula diferentes tipos de armazenamento (Local Storage, IndexDB, FileSystem e Session Strorage) disponíveis para uma aplicação web. O HCache aparece também na memória Cache acima das infraestruturas de armazenamento, para lidar com repetidas remissões. A política especifica a hierarquia de armazenamento. A hierarquia de armazenamento baseia-se no desempenho de camadas diferentes de armazenamento com armazenamento de alto desempenho no topo. Fonte: [65]

O ponto positivo do HCache foi a diminuição do tempo de carregamento das páginas com uso de técnicas voltadas na melhoria e aplicabilidade de Cache. Ao mesmo tempo, o ponto negativo concentra-se na limitação da ferramenta que não substitui diretamente o carregamento de imagem por tag estática, apenas

por carregamento definido por uma propriedade DOM (*Document Object Model*) ou por carregamento de uma API assíncrona explícita. Desta forma, páginas HTML com conteúdos estáticos não podem ser referenciados pelo HCache.

3.2.1 Discussão

Os trabalhos apresentados sobre Cache das aplicações envolvem o uso de Caches relacionados a dispositivos móveis na plataforma Android, mas nenhum deles remete o ideal sobre a análise de dados em Cache. A Tabela 3.2 consolida os pontos importantes observados sobre cada um deles.

Tabela 3.2: Discussão dos trabalhos relacionados com Cache

Questões	**Trabalhos**			
	DCC	**CACHÉ**	**CacheKeeper**	**HCache**
Funciona na plataforma Android?	Sim	Sim	Sim	Sim
Emprega Cache para Browser Mobile?	Não	Não	Não	Sim
Armazena dados em Cache?	Não	Sim	Sim	Sim
Limita Tamanho da Cache?	Não *	Não	Não	Não
Armazena dados em cartão externo?	Não	Sim	Não	Não
Padroniza formato de arquivo salvos em Cache?	Não **	Não	Não	Não
Emprega tempo médio de expiração?	Sim ***	Não	Não	Não
Aplica algum tipo de ofuscação dos dados?	Não	Não	Não	Não
Analisa o tipo de dado que está sendo salvo/removido?	Não **	Não	Não	Não

* Usa o tamanho da cache dos aplicativos como parâmetro para ser executado.
** Apenas remove da pasta de Cache os arquivos encontrados.
*** O tempo de expiração ocorre ao verificar se o aplicativo está ou não em uso.

De modo geral, os trabalhos possuem diferenças de uso, mecanismos e recursos, que são totalmente desproporcionais, olhando sob a ótica da quantidade de dados que podem ser armazenados, formato dos dados, memória, processamento e velocidade. Assim, semelhanças desses trabalhos com o proposto nesta dissertação não puderam ser encontradas. Por outro lado, os autores de todos os trabalhos apresentados enfatizam a necessidade de evitar que pessoas alheias capturem informações dispostas na Cache. Além disso, todos justificam que existe uma grande quantidade de aplicações que podem estar capturando dados e, de alguma forma, externando informações do usuário para cometimento de ilícitos a partir de dados coletados.

Apenas para contextualizar, é possível observar que nenhum dos trabalhos listados preocupa-se com o tamanho da Cache das aplicações. No caso do Caché, para obter uma grande quantidade de dados sobre uma região, pode ser necessário o armazenamento em cartão externo (aparentemente inseguro). Além disso, não há qualquer tipo de padronização de tamanho ou quantidade de dados a serem

salvos, bem como quanto tempo os dados irão permanecer disponíveis e como estão sendo salvos (formatos e visibilidade). Desta forma, pode-se enfatizar que não se sabe a quantidade de dados, tampouco o que existe salvo e se estes dados são sensíveis ou não, cabendo assim uma análise sobre estes dados que são salvos em cache pelas aplicações.

Capítulo 4

Análise de Caches

Este Capítulo é o cerne desta dissertação, pois apresenta todo o arcabouço proposto e implementado para análise de Caches na plataforma Android. Primeiramente, na Seção 4.1 é apresentado o modelo elaborado (adequado) para análise forense de Cache em dispositivos móveis, incluindo a discussão de cada uma das fases. Em seguida, na Seção 4.2, são apresentados os fatores para avaliação das implementações de Cache. Depois, na Seção 4.3, uma análise teórica (empírica) sobre as bibliotecas encontradas na literatura é feita, tomando como base os códigos fonte disponíveis, afim de verificar/constatar se os fatores de avaliação definidos são aplicados. Por fim, na Seção 4.4, através de um protótipo desenvolvido, experimentos são realizados e a análise dos resultados é explicada usando como referência o Modelo Forense proposto.

4.1 Modelo Proposto

Com a proliferação do crime digital no mundo, inúmeros modelos e procedimentos de Investigação Forense Digital foram ou estão sendo desenvolvidos. Na prática, cada país tende a desenvolver seus próprios procedimentos, alguns com foco no aspecto da tecnologia outros focados na porção de análise dos dados da investigação [40, 41].

Com base nos modelos apresentados na Tabela 2.4 (Seção 2.3), este trabalho optou por adequar o modelo CFSAP de McKemmish [42] para efetuar na análise de Cache. As razões para essa escolha são as seguintes:

- É um modelo adaptável, podendo ser desmembrado para atender determinadas necessidades;
- Abrange todos os aspectos necessários para análise da Cache, que apenas precisam ser modificados para atender aos objetivos deste trabalho;

- É um modelo muito utilizado, tendo servido como base para desenvolvimento de outros trabalhos (o modelo DFRWS [66], notoriamente é uma evolução do modelo de McKemmish).

A Figura 4.1 ilustra o processo conceitual proposto utilizando o modelo básico de McKemmish como base. Em seguida, cada uma das fases é descrita.

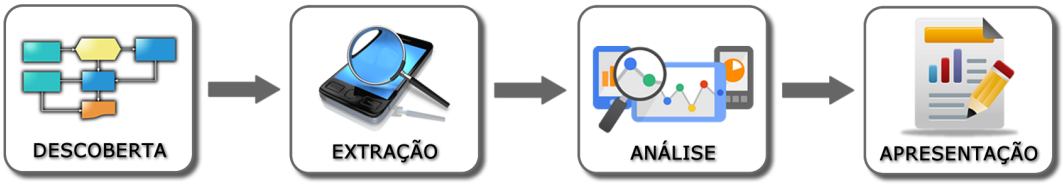

Figura 4.1: Modelo Conceitual Proposto

Descoberta

Esta fase é equivalente a etapa de identificação no modelo de McKemmish. Contudo, no escopo deste trabalho, esta fase se concentra em descobrir os dados existentes na cache. A diferença é que no modelo original, o foco é identificar o elemento criminoso, através da obtenção de provas. No caso da análise da Cache não existe tal necessidade, pois a Cache é considerada uma única evidência potencial.

Na prática, a fase de descoberta auxilia na identificação dos procedimentos necessários para análise. Esta fase se justifica em detrimento a variedade de Caches existentes, visto que cada aplicação pode fazer uso de uma ou mais bibliotecas de Caches.

Extração

A fase de extração é igual a do modelo de McKemmish (preservação) e consiste em extrair os dados da Cache para garantir sua originalidade e integridade. Nesta etapa, a extração das informações pode ser off-line ou em tempo real.

No processo em **tempo real**, os dados são extraídos com o dispositivo do usuário em funcionamento. Para que isso ocorra é necessária a existência de um mecanismo de suporte (uma aplicação no caso) no dispositivo, capaz de coletar e enviar os dados para uma local específico no dispositivo (uma pasta na memória física ou cartão de memória) ou para um serviço externo (por exemplo, uma cloud ou um serviço Web). Caso os dados extraídos fiquem no próprio dispositivo, as fases seguintes utilizarão recursos e memória do dispositivo. Caso contrário, os

dados serão analisados externamente e um feedback sobre a análise é retornado para a aplicação. É importante ressaltar que dados criados na Cache em tempo real podem ser deletados assim que a aplicação é encerrada ou na execução de um aplicativo de limpeza. Após salvar os dados como descrito, a aplicação deve garantir a originalidade como descrito a seguir no processo off-line.

No **off-line**, o dispositivo móvel a ser analisado deve ser conectado via USB, devendo o modo de depuração estar ativado. O processo de extração consiste em:

1. Duplicação, usando técnicas forenses;

2. Autenticação da cópia, feita com uso de algoritmos como o OWHF *One-Way Hash Function*, que permite a comparação com o original, evitando uma futura contestação sobre modificações na cópia.

Análise

Após os dados serem extraídos, a fase de análise deve ser realizada com objetivo de identificar elementos sensíveis, como os exemplos citados mais a frente neste Capítulo. Como visto na Seção 2.1.4, a Cache do Android está dividida em: **sistema**, que contém o Cache da máquina virtual Dalvik, e **aplicações**, onde o armazenamento ocorre na memória flash do dispositivo (Cache interna) e/ou no cartão de memória externo (Cache externa).

Pode se considerar esta fase do modelo como a mais importante, uma vez que:

- Os dados contidos na Cache externa não possuem atributos de segurança, diferente da Cache interna, que por políticas de segurança do Android, não permite que outras aplicações acessem dados de outra aplicação;

- Devido a variedade de formatos de Caches, um grande complicador é identificar cada tipo (caches de componentes, API de cache do Android, Caches personalizadas ou Caches de terceiros) para que seja possível identificar o vazamento de dados sensíveis do usuário;

- Algumas estruturas de Cache não possuem documentação ou estão incorretas. Geralmente existem bibliotecas que auxiliam o desenvolvedor, mas não existe uma documentação de forma pública da estrutura real da Cache relacionada a estas bibliotecas, o que dificulta ainda mais a análise.

Apresentação

Esta fase deve ser capaz de fornecer informações apresentáveis diante de um tribunal. A diferença é que deve conter a potencialidade dos dados sensíveis encontrados, categorizadas em:

- **Fail**, indica que os dados sensíveis foram recuperados a partir da análise da Cache do dispositivo.
- **Warn**, atribuído quando os dados em questão é descoberto, mas não coloca o usuário em risco.
- **Pass**, quando nenhum dado que está sendo procurado é localizado.

4.1.1 Aplicação do Modelo Proposto

Esta seção tem por finalidade apresentar como o modelo conceitual proposto é empregando-o na prática. Para tanto, é importante ressaltar que para realizar qualquer atividade ou experimento que leve em consideração o modelo proposto faz-se necessário assumir certas premissas relativas ao mundo real, especificamente sobre o dispositivo que será manipulado. São elas:

- **Dispositivo móvel em modo *root*** - normalmente, os dispositivos que executam Android são configurados de modo que o usuário não possua permissão total sobre o sistema por meio de acesso administrativo (*root*). Essa decisão de projeto resulta em um maior nível de segurança para a plataforma, semelhante ao que se preza em PCs, onde o usuário deve usar o mínimo de permissões necessárias para a realização de suas tarefas. Esse nível de segurança elevado é alcançado pelo fato do sistema impossibilitar que o usuário instale aplicativos com permissões de "superusuário", o que concederia tais permissões ao aplicativo em questão [16]. Com o dispositivo "rooteado", o pesquisador/investigador possui liberdade de acesso aos dados de aplicativos e do sistema.
- **Modo de desenvolvedor habilitado** - o recurso de "Opções de desenvolvedores" se torna indispensável para os desenvolvedores Android como um meio de acessar e controlar o seu dispositivo a partir do PC de forma rápida. Ao ativar tal modo, algumas opções são listadas: (i) habilitar a depuração por USB; (ii) capturar rapidamente relatórios de bugs de um dispositivo; (iii) mostrar o uso da CPU na tela; (iv) muitas outras opções para simular tensões de uma aplicação (app) entre outros recursos do tipo; (v) desenhar informações de depuração na tela, tais como limites de layout, atualizações em exibições GPU e outras informações.
- **Modo de depuração USB habilitado** - após habilitar o modo de desenvolvedor, deve-se habilitar a opção "Depuração por USB". Desta forma, o adbd (*Android Debug Bridge Daemon*) será iniciado com as permissões da shell de usuário. Esse depurador possibilita a instalação e desinstalação de

aplicativos, o gerenciamento de logs, a execução de comandos de shell, a cópia de arquivos para o dispositivo, a geração e restauração de backups, entre outros [16].

- **Armazenamento interno não criptografado** - por padrão os dispositivos móveis não são criptografados. Desta forma, temos a premissa que os dados do dispositivo a ser analisado "NÃO" estejam criptografados. Analisar dispositivos criptografados ainda é um dos maiores desafios (depende do algoritmo aplicado para cifrar os dados, pois alguns já foram quebrados) da atualidade das autoridades policiais e investigadores forenses.

Descoberta

Partindo do pressuposto que todas a premissas foram atendidas, a fase de descoberta do modelo consiste em identificar onde será realizada a análise. Neste caso, onde estão os dados de Cache das aplicações (memória e disco do próprio dispositivo). Assim, pode-se assumir que essa fase é simples e está diretamente ligada ao objetivo de analisar a Cache de todas as aplicações.

Extração

Na segunda fase (**Extração**), os dados são extraídos do dispositivo. Na prática, existem dois métodos para extração de dados (cópia/imagem). No primeiro é necessário que o dispositivo esteja "rooteado" (uma das premissas) e utilizar comandos da interface adb (*Android Debug Bridge*) para realização da cópia (imagem) da partição. O segundo método consiste em iniciar o dispositivo com imagens personalizadas, que são construídas especificamente para este propósito. Assim, por meio de um console de recuperação, o código desenvolvido extrai do dispositivo as informações necessárias. Uma vez que essa técnica emprega imagens criadas baseadas em versões do Android e do hardware do dispositivo e devido ao lançamento quase diário, tais ferramentas tornam-se complicadas, custosas e logo tornam-se obsoletas.

Análise

Na terceira fase do modelo (**Análise**), uma vez que esta fase compreende a análise sobre os dados existentes nas partições copiadas, fica a cargo do investigador usar a ferramenta que julgar necessário para análise dos dados. O que é preciso ilustrar é onde esses dados poderão ser encontrados. Em outras palavras, a partir de quais partições ou regiões identificadas a análise poderá ser realizada. O Ca-

che do Android está dividido em **Cache do Sistema** e **Cache dos Aplicativos**.

Em **Cache do Sistema**, a Dalvik, a máquina virtual utilizada pelo Android, mantém seu conteúdo em Bytecode (código de bytes em Java) na pasta */data/data/dalvik-cache*.
Já em **Cache dos Aplicativos** por padrão, os dados armazenados em Cache devem ser salvos na pasta */data/**pkg**/cache*, mas de fato, muitos desenvolvedores, por fazerem uso de bibliotecas de terceiros, não fazem ideia onde estas bibliotecas armazenam seus dados.

A seguir, os dados e formatos encontrados nesta análise na Cache dos aplicativos são descritos.

Cache Genérico

São dados de Cache encontrados na pasta */data/**pkg**/cache* das aplicações. A listagem a seguir mostra os arquivos com extensões desconhecidas encontrados no Android.

```
cache_bd.0
cache_bd.m
cache_its.m
cache_its_ter.m
cache_r.0
cache_r.m
cache_vts_<package>.0
cache_vts_<package>.m
cache_vts_inaka_<package>.0
cache_vts_inaka_<package>.m
cache_vts_labl_<package>.0
cache_vts_labl_<package>.m
```

Nestes arquivos constam estruturas com códigos de instrução de máquina (ex. $0x35$) e com informações de cabeçalho, data e hora, duração da URL dos dados em Cache, *Content-Type* dos dados e outros.

WebViewComponent Cache

Quando um aplicativo usa componente WebView para carregar páginas da Internet, as páginas e outros recursos relacionados a ela são armazenados no Cache privado de cada aplicação (app). Assim, esse Cache é composto de muitos arquivos, mas com apenas três formatos de arquivo. A Figura 4.2 ilustra esses arquivos.

Nos arquivos listados constam dados com notações de instruções de máquina, semelhante ao anterior, mas contendo informações de identificação do arquivo, quantidade de entradas armazenadas, total dos dados armazenados, último arquivo externo criado, flags, entre outras informações. Nos demais arquivos, existem dados brutos de HTML e JavaScript.

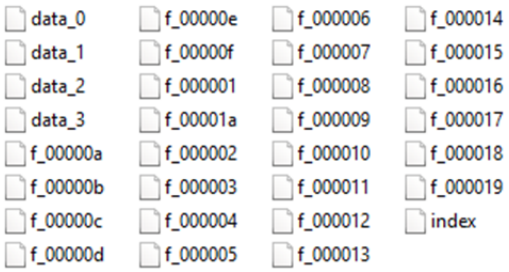

Figura 4.2: Lista de arquivos Cache WebView

Cache de Banco de Dados SQLite

O SQLite é uma biblioteca que implementa um banco de dados SQL embutido que é muito utilizado na plataforma Android por sua simplicidade e leveza. Muitas aplicações fazem uso (dispositivos e sistemas embarcados, ferramentas estatísticas e de análise, sítios com poucas requisições, etc.) e não necessita de instalação, configuração ou administração, gravando os dados em um único arquivo sem possuir dependências externas com outros servidores.

Galeria de imagens de Cache no padrão Android

A galeria de imagens de Cache utiliza um formato bem diferente dos demais. Ela cria três arquivos, onde o primeiro representa o índice e os outros dois são arquivos de dados (sendo que um permanece ativo e o outro inativo). As novas entradas são anexadas até atingir o limite do tamanho, fazendo com que o arquivo ativo e inativo sejam substituídos, onde o arquivo novo ativo é truncado e seu índice (tabela hash) também é limpo [67].

Objetos Java Serializados

Um objeto serializado é transformado em bytes, e pode ser armazenado em disco ou transmitido por um stream (objeto de transmissão de dados, onde um fluxo de dados serial é feito através de uma origem e de um destino) como ocorre com vídeos do YouTube, de modo que o objeto possa ser recriado a partir da cópia serializada em um momento posterior, e quando necessário.

Cache de arquivos DiskLRUCache

A biblioteca DiskLRUCache é utilizada para trabalhar com Cache em disco, aplicando a política de LRU (*Least Recently Used*). O arquivo criado em Cache usa uma estrutura para gerenciar as entradas, usando uma quantidade especificada

de espaço e cada entrada de Cache contém uma chave e um número fixo de valores [68], como mostrado a seguir:

```
libcore.io.DiskLruCache
1
100
2

CLEAN 3400330d1dfc7f3f7f4b8d4d803dfcf6 832 21054

DIRTY 335c4c6028171cfddfbaae1a9c313c52
CLEAN 335c4c6028171cfddfbaae1a9c313c52 3934 2342
REMOVE 335c4c6028171cfddfbaae1a9c313c52
DIRTY 1ab96a171faeeee38496d8b330771a7a
CLEAN 1ab96a171faeeee38496d8b330771a7a 1600 234
READ 335c4c6028171cfddfbaae1a9c313c52
READ 3400330d1dfc7f3f7f4b8d4d803dfcf6
```

Relatório

Na última fase do modelo (**Relatório**), esta deve ser capaz de fornecer informações apresentáveis diante de um tribunal, contendo a potencialidade dos dados sensíveis e ainda **categorizados** conforme visto na Seção 4.1.

Como demonstrado na fase de análise, o que podemos observar foi que os arquivos armazenados em Cache possuem diferenças como: (i) formato/estrutura; (ii) extensão e; (iii) tamanho, cabendo ainda um estudo mais aprofundado, capaz de nos fornecer subsídios para elaboração das potencialidades de forma categorizada.

Desta forma, a fase de relatório está contemplada nos resultados da Seção 4.4, mediante análise experimental de um protótipo que faz uso de algumas bibliotecas de Cache descritas na Seção 2.2 do Capítulo 2, cuja análise é realizada sob a ótica de fatores de avaliação direcionados à Cache.

4.2 Fatores de Avaliação

Em todas pesquisas e revisões na literatura e até onde se tem conhecimento, não existe um conjunto bem definido de fatores destinados a exprimir os principais pontos a serem considerados durante a avaliação de Cache, especialmente em dispositivos móveis. Desta forma, buscou-se como contribuição estabelecer tal conjunto de fatores.

Os fatores aqui propostos são focados em avaliar a segurança e são derivadas da análise de recomendações encontradas na literatura sobre segurança, especialmente sobre segurança móvel da OWASP [69]. Embora os fatores não sejam de fato métricas, é factível acreditar que eles constituem valiosa ferramenta no

processo de análise dos benefícios e limitações inerentes as bibliotecas Cache em Android.

Os fatores considerados relevantes são:

1. Criptografia dos Dados;
2. Tamanho Máximo da Cache;
3. Tempo de Expiração.

A seguir, cada um deles é detalhado.

4.2.1 Criptografia dos Dados

Dispositivos móveis podem conter dados sensíveis do usuário que não devem parar nas mãos de terceiros. Como tais dados podem estar disponíveis na Cache, tornam-se alvos fáceis para pessoas mal intencionadas (no caso de perda do dispositivo) ou por processos ocultos. Seguindo uma recomendação da OWASP [69], o emprego de práticas de criptografia é fundamental para evitar que dados, possivelmente sensíveis, sejam vistos ou capturados por outras aplicações ou terceiros sem autorização.

Assim, esta dissertação propõe a verificação da existência de mecanismos para criptografar os dados a serem salvos em Cache como um fator. Ao analisar o funcionamento da biblioteca, sugere-se indicar se a mesma possui ou não algum tipo de criptografia para os dados em Cache.

4.2.2 Tamanho Máximo da Cache

Embora já existam dispositivos móveis com grande capacidade de armazenamento interno, externo, bem como em memória (32, 64, 128, .. GB), é preciso considerar que os custos ainda são elevados para grande parte dos usuários. Desta forma, é importante que qualquer aplicativo que faça o melhor uso de armazenamento de dados, levando em consideração o tamanho máximo da Cache. Mesmo que questionamentos sobre a possibilidade de expansão do espaço de armazenamento através de cartões externos possam ser feitos para armazenar grandes quantidade de dados, também é preciso assumir que não existem garantias de que informações sensíveis não serão ou não ficarão gravadas neste espaço, o que acaba por impactar na segurança do dispositivo e do usuário.

Na página de desenvolvedores Android [13], existe uma sugestão para que os dados armazenados em Cache não ultrapassem 1MB. Já alguns autores e desenvolvedores [70] recomendam que seja entre 5 e 7 MB para WebApp, dependendo do caso. Desta forma, o tamanho máximo da Cache é um fator válido para este

trabalho e deve analisar o funcionamento das bibliotecas sobre sua capacidade de limitar ou não o tamanho máximo da Cache.

4.2.3 Tempo de Expiração

Como explicado durante esta dissertação, Cache é um conceito extremamente útil para melhorar o desempenho de um sistema. Uma estratégia de segurança e desempenho relacionada a função da Cache é estipular o tempo que o conteúdo (objetos) será mantido. O fator Tempo de Expiração tem a função de definir por quanto tempo um determinado dado ficará ou estará disponível para acesso na Cache.

É importante notar que esse tempo depende do uso e da aplicação. Em alguns casos onde, por exemplo, quando o conteúdo é formado por dados absolutos, pode ser útil manter os dados em Cache por um longo período de tempo. Assim, manter os dados durante minutos, horas, dias e até semanas é mais do que aceitável. Já em outros casos, como quando o conteúdo é mais dinâmico, o tempo de permanência na Cache é mais restrito, variando de segundos até poucos minutos.

Em decorrência da existência dessa ambiguidade de tempo para os dados em Cache, este trabalho propõe a verificação da existência de mecanismos de tempo para expiração do conteúdo armazenado em Cache pelas bibliotecas como um fator de avaliação. Ao analisar o funcionamento da biblioteca, sugere-se indicar se a mesma possui ou não algum tipo de tempo de expiração para os dados em Cache e, se possível, informar qual(is) é(são) esse(s) tempo(s).

4.3 Análise Empírica

Esta análise empírica (teórica) destinou-se a comprovar duas conjecturas primordiais para este trabalho:

1. Demonstrar que, embora desenvolvidas com a finalidade de melhorar o desempenho, as bibliotecas de Cache para Android não estão tão preocupadas com isso;

2. Confirmar que aspectos de segurança são esquecidos e/ou negligenciados.

Assim, como "prova de conceito", cada um dos três fatores elaborados para bibliotecas de Cache Android foram aplicados e o resultado é sumarizado na Tabela 4.1.

Desta forma, a discussão dos pontos apresentados serão analisados sob o aspecto de cada Fator de Avaliação (Criptografia, Tamanho Máximo da Cache e Tempo de Expiração).

Tabela 4.1: Análise das Bibliotecas de Cache para Android sob a Ótica dos 3 Fatores

Biblioteca	**Métricas**		
	Criptografia	**Tamanho Máximo**	**Tempo de Expiração**
LRUCache	Não implementa	É implementado um padrão no construtor ($intcacheSize = 4 * 1024 * 1024$) igual a 4 MB	Implementa a classe *TimeExpiringLruCache* com Tempo limite em milissegundos
DiskLRUCache	Não implementa	Utiliza o parâmetro $maxSize$ na classe *DiskLruCache* para que seja especificado pelo desenvolvedor o tamanho inicial.	Não implementa.
Android Easy Cache	Não implementa	Implementa em RAM, em Disco ou em ambas, mas o desenvolvedor especificar o valor inicial.	Como usa a LRUCache, utiliza a *TimeExpiringLruCache*. Não implementa para disco.
Simple DiskCache	Não implementa	igual ao da biblioteca DiskLRUCache	Não implementa
HttpResponse Cache	Não implementa	Deve ser especificado pelo desenvolvedor	Não implementa
ObjectCache	Não implementa	Pode ser implementado pelo desenvolvedor, mas por padrão adota 10 MB.	Implementa a classe *CacheManager* com a função $ExpiryTimes$ que especifica o tempo com escalas já definidas (segundos, minutos, horas, dias, semanas, meses e anos)
Qachee	Não implementa	Usa por padrão 1/8 da memória disponível (*final int cacheSize = maxMemory/8;*)	Padrão de 1 minuto, mas o desenvolvedor pode escolher.
Reservoir	Não implementa	Tem que ser especificado pelo desenvolvedor.	Não implementa
Android BitmapCache	Não implementa	Padrão adotado é 3 Mb, para RAM, e 10 Mb para disco, mas o desenvolvedor pode escolher	Como usa a LRUCache, utiliza a *TimeExpiringLruCache*. Não implementa para disco.
Kinvey	Implementa	Em memória, o padrão é 16 Kb. Em Disco, armazena usando SQLite3 em modo normal e off-line.	Como usa a LRUCache, utiliza a classe *TimeExpiringLruCache*. Não implementa para disco
Expirable DiskLruCache	Implementa, usa a biblioteca Conceal [37]	Deve ser inicializado pelo desenvolvedor com valores em bytes	Implementa uma função $evictionTimeSpan$ onde o tempo de expiração é passado em milissegundos. É opcional
Carbonite	Não implementa	Padrão adotado é de 256 Kb para memória RAM e 10 MB para disco	Como usa a LRUCache, utiliza a *TimeExpiringLruCache*. Não implementa para disco

Criptografia

Como visto na Tabela 4.1, apenas duas bibliotecas permitem o uso de criptografia para proteger os dados em Cache.

A biblioteca **Expirable DiskLRUCache** de fato não implementa criptografia. Na verdade, faz uso da biblioteca Conceal (Facebook) para executar a criptografia no Android. A Conceal foi projetada para ser capaz de criptografar grandes arquivos no disco de uma maneira rápida e eficiente. Usa algoritmos criptográficos específicos como *AES-GCM* (*Advanced Encryption Standard* com *Galois/Counter Mode*) e HMAC-SHA1 (*Hash-based Message Authentication Code*), ambos empregados tanto para verificar a integridade quanto a autenticidade de uma mensagem.

Já a biblioteca **Kinvey** possui uma extensão de criptografia implementada usando *AES/CBC/PKCS5Padding* (*Advanced Encryption Standard*) aplicando modos de cifragem com *Cipher-Block Chaining* (CBC) e *PKCS5Padding*. Contudo, o algoritmo *AES/CBC/PKCS5Padding* provê apenas confidencialidade, mas não assegura a *integridade* da mensagem [71]. Além disso, para garantir o armazenamento de dados off-line no SQLite, a Kinvey faz uso de uma extensão de

criptografia baseada no *SqlCipher* [1], que emprega também o modelo de cifragem CBC com HMAC_SHA1 (*Hash-based Message Authentication Code*) e PBKDF2 (*Password-Based Key Derivation Function* 2) [73].

Analisando a aplicabilidade dos algoritmos de criptografia, e não as bibliotecas de Cache, pôde-se observar que a biblioteca do Conceal aparenta ser melhor, pois oferece garantia de confidencialidade e integridade, enquanto a Kinvey garante apenas confidencialidade.

Tamanho Máximo da Cache

Como forma de melhor explicar o Tamanho da Cache, a Tabela 4.2 descreve as características de cada uma das bibliotecas analisadas.

Tabela 4.2: Tamanho da Cache nas bibliotecas

Biblioteca	**Tipo**		**Tamanho Inicial**	
	RAM	**Disco**	**RAM**	**Disco**
LRUCache	✓		4Mib***	
DiskLRUCache	*	✓		**
Android Easy Cache	✓	✓	**	**
Simple DiskCache	*	✓		**
HttpResponse Cache		✓		10 MiB ***
Qachee	✓		****	
Reservoir		✓	**	
Android BitmapCache	✓	✓	3MB***	10MB***#
ObjectCache	##	✓		10MB***
Kinvey	✓	✓	**	###
Expirable DisKLRUCache		✓		**
Carbonite	✓	✓	**	***

* Usa 8MB para Buffer I/O
** O desenvolvedor é obrigado a especificar um tamanho inicial
*** Por padrão possui um tamanho inicial que pode ser alterado
**** Usa 1/8 do total da memória do dispositivo
# Por padrão o uso de cache em Disco vem desabilitado
Usa cache de tempo de execução na memória de seus objetos
Quando o modo offline está habilitado, os dados são armazenados no DB SQLite

Percebe-se na Tabela 4.2 que existem bibliotecas que trabalham em memória, disco ou ambos os modos. Existem seis bibliotecas que trabalham com memória (LRUCache, Android EasyCache, Qachee, Android BitmapCache, Kinvey e Carbonite), mas algumas delas (DiskLRUCache, Simple DiskCache e ObjectCache)

[1]SQLCipher é uma extensão open source para SQLite que fornece criptografia transparente AES de 256 bits de arquivos de banco de dados. Fonte: [72]

trabalham com objetos em memória antes de serem armazenadas na Cache de disco. A **DiskLRUCache**, apesar de salvar dados em disco, usa 8 MB como *buffer* de entrada e saída (I/O). A **Simple DiskCache**, por herdar as mesmas características da DiskLRUCache, segue o mesmo padrão. A **ObjectCache** também salva os dados em disco (usando a biblioteca DiskLruCache) com representações JSON de seus objetos (usando GSON) e um Cache de tempo de execução na memória de seus objetos.

Já as bibliotecas de armazenamento em disco (DiskLRUCache, Android EasyCache, Simple DiskCache, Http ResponseCache, Reservoir, Android BitmapCache, ObjectCache, Kinvey, Expirable e Carbonite) possuem padrões de inicialização de tamanho já definidas (HttpResponse Cache, Android BitmapCache, ObjectCache e Carbonite) ou obrigam o desenvolvedor a definir um tamanho inicial para Cache em disco (DiskLRUCache, Android Easy Cache, Simple DiskCache e Expirable DisKLRUCache). A biblioteca **Android BitmapCache** armazena dados em ambos os modos, mas, por padrão, deixa o armazenamento de Cache em disco desabilitado. Já a biblioteca **Kinvey**, quando no modo *offline*, permite que os dados sejam armazenados no banco de dados SQLite.

Dentre as 12 (doze) bibliotecas apresentadas, observa-se que a **Qachee** aplica uma regra interessante e diferente das demais. Ela calcula o uso de memória pegando o total da memória do dispositivo dividido por oito, dispensando a necessidade do desenvolvedor especificar um valor inicial para armazenamento de Cache em memória, como visto no código abaixo:

```
// construtor privado
// impede instanciação de outras classes
private QacheeManager() {
        final int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);
        // Usa 1/8 da memória disponível para memória cache
        final int cacheSize = maxMemory / 8;

        qachee = new LruCache<>(cacheSize);
        expirationTime = ExpirationTime.ONE_MINUTE;
}
```

Tempo de Expiração

Quando uma aplicação faz uso da Cache em memória ou disco, os dados armazenado deveriam deixar de existir com o encerramento da aplicação ou com o uso do coletor de lixo da máquina virtual(*garbage collector*). Esse aspecto, chamado aqui de Tempo de Expiração, serve para remover algum objeto e dar lugar a outro em detrimento ao tamanho (armazenamento em bytes) da pasta de Cache. A Tabela 4.3 apresenta uma análise das bibliotecas sob a ótica do fator Tempo de Expiração.

Das bibliotecas analisadas, 9 (nove) usam a política LRU (*Least Recently*

Tabela 4.3: Tempo de Expiração das Bibliotecas

Biblioteca	Tipo		Tempo de Expiração	
	RAM	Disco	RAM	Disco
LRUCache	✓		*	
DiskLRUCache		✓		*
Android Easy Cache	✓	✓	*	*
Simple DiskCache		✓		*
HttpResponse Cache		✓		*
Qachee	✓		**	
Reservoir		✓		*
Android BitmapCache	✓	✓	*	*
ObjectCache	#	✓		***
Kinvey	✓	✓	*	****
Expirable DisKLRUCache		✓		*****
Carbonite	✓	✓	*	##

* Usa a política de substituição de arquivos com LRU
** Define Política de Expiração de Tempo com "ExpirationTime.ONE_MINUTE"
Opções: TEN_SECONDS, THIRTY_SECONDS, ONE_MINUTE,
TWO_MINUTES, TEN_MINUTES, TWENTY_MINUTES
*** Define Política de Tempo de Expiração com "expiryTimeSeconds"
Opções: ONE_SECOND(1), ONE_MINUTE(60), ONE_HOUR(60 * 60),
ONE_DAY(60 * 60 * 24), ONE_WEEK(60 * 60 * 24 * 7),
ONE_MONTH(60 * 60 * 24 * 30), ONE_YEAR(60 * 60 * 24 * 365);
**** Caso a opção de dados estiver offline, o tempo de permanência dos dados
será até o dispositivo se conectar a internet
***** Define uma política de Tempo de Expiração com "evictionTimeSpan"
# Usa cache de tempo de execução na memória de seus objetos
Faz verificação de tempo com "TimeUnit"

Used) como forma de Tempo de Expiração. São elas: LRUCache, DiskLRUCache, Android EasyCache, Simple DiskCache, HttpResponseCache, Reservoir, Android BitmapCache, Kinvey e Carbonite. Dentre essas, 5 (cinco) - Android EasyCache, Android BitmapCache, ObjectCache, Kinvey e Carbonite - trabalham com tempo de expiração tanto em memória quanto em disco. Vale ressaltar que na ObjectCache e na Carbonite, o uso da memória serve para dar celeridade ao processo de uso de objetos (JSON e POJO's) para, então, armazenar em disco.

Contudo, algumas bibliotecas aplicam seu próprio Tempo de Expiração. A **Qachee** define políticas de tempo de expiração em memória com aplicação de um parâmetro *ExpirationTime*, com opções de tempo que variam entre dez segundos a vinte minutos. A **ObjectCache** também define uma política de tempo de expiração através do parâmetro *expiryTimeSeconds*, com opções que variam entre um segundo a um ano. A **Kinvey**, por fazer uso da biblioteca LRUCache,

aplica apenas a política de LRU (memória) e faz armazenamento de dados offline em Cache (disco) num banco de dados SQLite 3 e, que, por meio de políticas de Cache podem sincronizar dados, em caso da existência de uma rede de dados. A **Expirable DisKLRUCache** define uma política de tempo de expiração através do parâmetro *evictionTimeSpan* e a **Carbonite**, apesar de usar POJOs em memória, faz o controle de tempo aplicando a política de LRU somada a uma verificação de tempo com *TimeUnit*.

4.4 Análise Experimental

Uma vez que a análise empírica baseada nos fatores foi realizada, torna-se necessário realizar uma análise via experimentação. Para tanto, um protótipo funcional foi desenvolvido para testar as bibliotecas de acordo com os fatores de avaliação e medir seu desempenho, bem como fazer uso do Modelo proposto na Seção 4.1 do Capítulo 4.

4.4.1 Protocolo Experimental

O ambiente de desenvolvimento da aplicação utilizou a plataforma Android Studio (versão 1.2), a linguagem Java, PHP (versão 5.5.26) e MySQL (versão 5.7.6). O hardware empregado na elaboração foi uma estação de trabalho Intel Core 7 de 3.4 Ghz, com 8 GB de memória RAM, disco de 500 GB e plataforma Linux, distribuição Ubuntu 14.04.

Todos os experimentos foram monitorados utilizando o *AndroidDevice Monitor*, onde é possível se ter controle dos diretórios do sistema operacional, incluindo o diretório (*/data/data/<**pkg**>/cache*).

Embora 12 (doze) bibliotecas tenham sido avaliadas na análise empírica, para os testes práticos apenas as seis que implementam Cache em disco foram analisadas. Como forma de melhor avaliá-las, os experimentos foram realizados gerando Cache de imagens e de texto. A seguir cada um deles é detalhado.

4.4.2 Cache de Imagens

Para avaliar a Cache de imagens foi utilizado um conjunto de 60 figuras extraídas do site *tumblr*, todas variando entre 400 e 600 pixels. O protótipo utilizou as bibliotecas Android-EasyCache, SimpleDiskCache e Android-BitmapCache, todas avaliadas seguindo os fatores apresentados na Seção 4.2, considerando a praticidade da implementação. A Tabela 4.4 resume essa avaliação.

Percebe-se na Tabela 4.4 que as bibliotecas de imagem analisadas não fazem uso de qualquer tipo de criptografia, mas todas aplicam algum tipo de

Tabela 4.4: Análise experimental de bibliotecas de Cache (imagem) sobre os fatores de avaliação

Características Analisadas	**Bibliotecas**		
	Android EasyCache	**Simple DiskCache**	**Android BitmapCache**
Aplica Criptografia / Ofuscação?	Ofuscação	Ofuscação	Ofuscação *
Pode-se visualizar o conteúdo do arquivo?	Sim	Sim	Sim
Arquivo armazenado na pasta do app?	Sim	Sim	Não
Tamanho do arquivo difere do orignal?	Sim	Sim	Sim
Extensão do arquivo	".0"ou ".1"	".0"ou ".1"	".0"ou ".1"

* Abrindo o arquivo em forma textual verifica-se um texto com vários caracteres especiais, onde algumas passagens no texto podemos observar o local do emissor da imagem.

ofuscação sobre nome dos arquivos armazenados. Por exemplo, a figura *tumblr_lht5uy6khS1qed3e3o1_500.jpg* (extraída do *tumblr*) é representada pelo nome *24c52ace61c49sc2ccf4f218a608dee9.0* na biblioteca Android-BitmapCache e *4651ed93779e029e9146a3011578571.0* nas bibliotecas Android-EasyCache e Simple-DiskCache.

Vale ressaltar que: (i) todas as bibliotecas salvam os arquivos de imagem com a extensão .0 ou .1; (ii) as bibliotecas Android-EasyCache e SimpleDiskCache, por derivarem do projeto original da DiskLRUCache, geram o mesmo nome ofuscado; (iii) apesar da ofuscação, o conteúdo das imagens armazenado pelas bibliotecas pode ser visualizado.

O apêndice A apresenta o nome ofuscado das imagens utilizadas no experimento por cada uma das bibliotecas.

A experimentação também mostrou que a biblioteca Android BitmapCache, ao salvar o arquivo, insere algumas informações que identificam o emissor da imagem e que ela não armazena seus dados no caminho padrão de Cache da aplicação. Além disso, em todas as bibliotecas o tamanho do arquivo se difere do original, possuindo um arquivo final muitas vezes bem maior que o original.

Para finalizar, avaliando as bibliotecas mediante a última fase do Modelo Proposto na Seção 4.1, pode-se afirmar que todas as bibliotecas avaliadas podem ser classificadas como "*Fail*", uma vez que dados sensíveis foram recuperados a partir da análise da Cache do dispositivo.

4.4.3 Cache de Texto

Para avaliar a Cache de texto, o protótipo funcional apresentava um formulário contendo dados sensíveis como nome e senha (em um formulário de login), bem como outros dados como e-mail, documentos pessoais (CPF, RG), endereço e etc. Neste experimento, foram empregadas as bibliotecas Android EasyCache, HttpResponseCache, Expirable DiskLRUCache e Reservoir.

A Tabela 4.5 resume algumas das características avaliadas em detrimento aos

fatores de avaliação.

Tabela 4.5: Análise experimental de bibliotecas de Cache (texto) sobre os fatores de avaliação

Características Analisadas	**Bibliotecas**			
	Android EasyCache	**HttpResponseCache**	**Expirable DiskLRUCache**	**Reservoir**
Aplica Criptografia / Ofuscação?	Ofuscação *	Ofuscação **	Ambos	Ofuscação
Pode-se visualizar o conteúdo do arquivo?	Sim	Sim	Não	Sim
Arquivo armazenado na pasta do app?	Sim	Sim	Sim	Sim
Tamanho do arquivo difere do orignal?	Sim	Não se aplica	Não se aplica	Não se aplica
Extensão do arquivo	".0"ou ".1"	Sem extensão ***	".0"ou ".1"	".0"ou ".1"

* Pode-se visualizar também conteúdos sensíveis do usuário, por exemplo, senhas.
** O arquivo contem algumas informações de serviços e protocolos.
*** Os arquivos em cache desta biblioteca não possuem extensão, apenas possuem a ofuscação no seu nome.

Percebe-se na Tabela 4.5 que as bibliotecas Android EasyCache, HttpResponseCache e Reservoir aplicam algum tipo de ofuscação em seus arquivos, mas seu conteúdo pode ser visualizado, inclusive dados sensíveis do usuário podem ser visualizados (ex. senhas) em texto plano. Somente a biblioteca ExpirableDiskCacheLRU, além de aplicar ofuscação, criptografa o arquivo, impedindo que seu conteúdo seja visualizado. Como já mencionado, ela faz uso da biblioteca Conceal para isso.

Além disso, todas as bibliotecas armazenam seus arquivos na pasta padrão do aplicativo. Quanto a extensão, todas usam o padrão “.0” ou “.1”, exceto a HttpResponseCache que não possui extensão.

Para finalizar, avaliando as bibliotecas mediante a última fase do Modelo Proposto na Seção 4.1, pode-se afirmar que:

1. A biblioteca Android Easy Cache é classificada como "*Fail*", uma vez que dados sensíveis foram recuperados a partir da análise da Cache do dispositivo.

2. As bibliotecas HttpResponseCache e Reservoir foram classificadas como "*Warn*", pois embora dados possam ser descobertos, nenhum é vizualizado.

3. A biblioteca ExpirableDisKLRUCache foi classificada como "*Pass*", pois nenhum dado procurado foi localizado. Esta biblioteca ofusca e criptografa seus dados armazenados, buscando impedir que seu conteúdo seja visualizado por terceiros.

Capítulo 5

Recomendações para Cache

Este Capítulo apresenta recomendações de segurança baseadas nas principais empresas de desenvolvimento de ferramentas e/ou aplicativos, bem como organizações de segurança da informação. A ideia deste Capítulo é mostrar como impedir que terceiros possam, através das falhas, encontrar brechas e com isso tirar proveito e/ou prejudicar o usuário. Uma discussão é feita no final e alguns apontamentos são realizados.

5.1 Google

A Google oferece, em sua página de desenvolvedores, sugestões voltadas à segurança no desenvolvimento de aplicações. Por exemplo, ela sugere e limita o tamanho máximo de Cache a 1MB [13]. Também sugere o uso de criptografia para dar garantias de segurança aos dados do usuário bem como o não salvamento de informações confidenciais em armazenamento externo [74].

Como as sugestões da Google não remetem apenas a estes três exemplos, há um tópico na página chamado de *Best Practices for Security & Privacy*, que contém artigos com informações sobre como manter os dados do aplicativo seguro. Na página é ressaltado a importância do desenvolvedor estar familiarizado com as melhores práticas de segurança Android, relacionando artigos como:

- Secure tips (Dicas seguras) - Como executar várias tarefas e manter os dados de seu aplicativo e dados do usuário seguro.
- Security with HTTPS and SSL (Segurança com HTTPS e SSL) - Como garantir que a aplicação é segura ao realizar transações de rede.
- Updating Your Security Provider to Protect Against SSL Exploits (Atualizando seu provedor de segurança para proteger contra Exploits SSL) -

Como usar as atualizações do Google Play (provedor de segurança) para proteger contra exploits SSL.

- Checking Device Compatibility with SafetyNet (Verificar compatibilidade do dispositivo com SafetyNet) - Como usar o serviço SafetyNet para analisar um dispositivo onde o aplicativo está sendo executado e obter informações sobre a sua compatibilidade.

- Enhancing Security With Device Management Policies (Reforço da segurança das políticas de gerenciamento de dispositivos) - Como criar um aplicativo que aplica políticas de segurança nos dispositivos.

5.2 OWASP

A Open Web Application Security Project (OWASP) é uma comunidade online dedicada à segurança de aplicações Web. Ela oferece uma série de informações e ferramentas em temas como: (i) Princípios de codificação segura; (ii) Bibliotecas de programação segura, envolvendo aspectos como validação de HTML e CSS em várias linguagens de programação; (iii) Guia de revisão de código para identificar vulnerabilidades de estouro de buffers, injeção de código, validação de dados, cross-site scripting, cross-site request forgery, logging issues, integridade de sessões e condições de corrida (race conditions); (iv) Melhores práticas de codificação segura e guias em linguagens e plataformas .NET; (v) Guia de teste de software para segurança; (vi) entre outros.

Dentre as várias publicações que OWASP disponibiliza e gere, a *OWASP Mobile Security Project* categoriza, a cada ano, as dez vulnerabilidades mais predominantes em aplicações Web, abordando-se algumas dessas vulnerabilidades e ideias referentes a prevenções, do ano de 2014, mas já voltadas aos dispositivos móveis.

Armazenamento Inseguro dos Dados (Insecure Data Storage)

Refere-se a perda ou vulnerabilidade dos dados do usuário, seja por exposição de dados confidenciais, vulnerabilidade de logs ou históricos de transações, ou até mesmo roubo/perda do dispositivo.

Uma vez que a regra básica de desenvolvimento para dispositivo móveis é não guardar dados a não ser que seja absolutamente necessário, prevenções para diminuir esse tipo de vulnerabilidades são:

- Não usar áreas públicas para armazenar dados (cartão SD);

- Caso for armazenar dados localmente usar a API do Android "*setStorageEncryption*";

- Para armazenar dados em cartão externo, usar a biblioteca "*javaz.crypto*" que garante certo nível de segurança e;
- Qualquer propriedade do *SharedPreferences* não deve estar em modo *MODE_WORLD_READABLE* (modo que habilita a troca de informações entre aplicativos).

Vazamento Não Intencional de Dados (Unintended Data Leakage)

Refere-se ao vazamento não-intencionado de dados que acontece quando o desenvolvedor, sem saber, coloca dados sensíveis num local do dispositivo móvel que é facilmente acessado por outros aplicativos. Isto inclui vulnerabilidades do sistema operacional, *frameworks*, hardware, entre outros.

Como soluções para esse tipo de vulnerabilidade, a OWASP recomenda:

- Ter cuidado com diretórios/arquivos temporários;
- Fazer debug antes de lançar uma aplicação (app) no mercado para verificar arquivos que são criados ou modificados;
- Verifique bibliotecas de terceiros e os dados que ela utiliza ou se elas guardam algum dado sensível do usuário e;
- Teste sua aplicação em várias plataformas diferentes.

Autorização e Autenticação Fracas (Poor Authorization and Authentication)

Também pode ser encontrado em aplicações Web, mas se difere um pouco quando ocorre em dispositivos móveis. Por exemplo, em aplicações Web é esperado que o usuário esteja online, autenticado e que sua sessão de conexão seja estável para ter acesso contínuo à Internet. Já no âmbito de aplicações móveis isso não é verdade. Conexões com a Internet em celulares e tablets são menos confiáveis ou previsíveis, quando comparadas às conexões tradicionais Web. Afinal, a qualquer momento o usuário pode perder o sinal de sua internet 3G ou de seu WiFi.

Neste sentido, as recomendações apontadas são:

- Force o usuário a não usar senhas fracas;
- Não usar somente o device ID na autenticação;
- Quando possível, certifique-se que todas requisições de autenticação aconteçam no lado servidor. Em caso de autenticação com sucesso, dados da aplicação serão carregados no dispositivo. Isso irá garantir que dados da aplicação só estarão disponíveis depois de uma autenticação bem sucedida;

- Funcionalidade de autenticação persistente (feature de Remember Me) nunca deve guardar a senha do usuário no dispositivo;
- Se o armazenamento de dados no lado cliente for mesmo necessário, os dados devem ser encriptados usando uma chave de encriptação derivada a partir das credenciais de login do usuário. Desta forma, haverá uma garantia que os dados guardados só serão acessíveis ao entrar com as credenciais corretas.

Quebra de Criptografa (Broken Cryptography)

Relata problemas com criptografia de dados, onde uma pessoa pode ter acesso físico aos dados de uma aplicação. Ocorre quando a criptografia usada é fraca demais ou existe algum software malicioso no dispositivo enviando informações para um agente de ameaça ou até mesmo algum *sniffer* capturando o tráfego de informações.

Como medidas de prevenção, a OWASP cita:

- Nunca salvar credenciais ou qualquer dado que seja de extrema importância;
- Desativar registro de teclas por campo;
- Utilizar diretivas anti-cache para utilização de conteúdo web;
- Uso do algoritmo SHA-256;
- Não utilizar os algoritmos RC2, MD4, MD5 ou SHA1.

5.3 Discussão

De forma mais abrangente que a abordagem da OWASP, a norma ISO/IEC 17799:2005 apresenta (em sua seção 12) uma lista de recomendações e práticas relacionadas à segurança na aquisição, desenvolvimento e manutenção de sistemas de informação. A maioria dessas recomendações é relacionada com a busca por software seguro [75].

Percebe-se que as recomendações de segurança são voltadas de modo geral para o desenvolvimento de aplicações, sendo ela para computadores (local ou Web) ou dispositivos móveis. Não há no entanto, recomendações voltadas especificamente para a construção de bibliotecas que fazem uso de Cache, mas que é possível construir partindo dos princípios de segurança descritos, buscando adaptar estes conceitos e aplicar em conformidade com as necessidades, buscando garantias sobre os dados armazenados ou manipulados pelas bibliotecas de Cache.

Partindo disso, pode-se elencar fatores considerados importantes para a construção de bibliotecas de Cache para uso das aplicações móveis como:

- Uso da RFC 2616 como base, pois apesar de referenciar apenas Caching HTTP, contém especificações que podem ser empregadas como políticas e diretivas para Cache;
- Seguir a recomendação de medidas de prevenção "M2, M4, M5 e M6" do OWASP Mobile Security Project;
- Dados privativos do usuário devem ser tratados com alta prioridade, buscando mecanismos de ofuscação contra ataques ou códigos maliciosos, cuja finalidade é o vazamento de dados para fins em sua maioria ilícitos;
- Que o tempo de permanência dos dados armazenados tenham tempo de expiração definido e com possibilidade de customização;
- Que o tamanho da Cache tanto para disco como para memória sejam customizados obedecendo às recomendações da plataforma ou normas voltadas para este propósito;
- Que diretórios criados para armazenar dados relacionados à Cache, sejam padronizados, com caminho do aplicativo / nome da biblioteca de Cache utilizada;
- Que os arquivos disponibilizados sigam extensões padronizadas de arquivos (estrutura de dados), podendo o desenvolvedor customizar, escolhendo e aplicando regras de segurança para os dados a serem armazenados em conformidade com a importância e confidencialidade;
- Que numa autenticação (login), uma biblioteca de Cache nunca permita armazenar dados do usuário ou mesmo login e senha;

Capítulo 6

Conclusões

Neste Capítulo, a dissertação é concluída reiterando-se as contribuições oferecidas por este trabalho e ainda apontando os trabalhos futuros que podem dar prosseguimento a este projeto de pesquisa. Na Seção 6.1, as considerações finais são feitas com foco nos objetivos que nortearam o desenvolvimento deste trabalho. Em seguida, as seções de dificuldades encontradas e trabalhos discorrem sobre problemas no desenvolvimento deste trabalho e das oportunidades de melhorias identificadas no transcorrer do projeto, que podem ensejar novos trabalhos de pesquisa.

6.1 Considerações Finais

Esta dissertação apresentou uma investigação sobre bibliotecas que implementam cache de aplicação na plataforma Android, por meio da definição de um modelo forense de análise e experimentação prática, visando comprovar o mau uso e/ou descaso no âmbito da segurança da informação, especialmente a informações sensíveis e/ou confidenciais, tendo como argumentos que: (i) dados sensíveis precisam ser protegidos (com criptografia, por exemplo) [16]; (ii) levando em consideração que quase todas as bibliotecas avaliadas não aplicam mecanismos de segurança; (iii) não existem pesquisas e, por enquanto, grandes interesses em tentar padronizar esquemas de segurança para Caches.

Além disso, todos os questionamentos sobre o uso da Cache em Android apresentadas no começo desta dissertação foram ou podem ser respondidos:

1. Não existem regra, padrão ou práticas de uso que sejam seguidas no desenvolvimento de aplicações, tanto puras quanto as que fazem uso de bibliotecas que implementam ou re-implementam Cache;

2. A plataforma Android não aplica ou define normas no desenvolvimento e

uso de Caches. O que existe são recomendações de segurança na página do desenvolvedor que enfatizam como fazer o armazenamento seguro dos dados do usuário e trata sobre a privacidade. Cabe ressaltar que não há um artigo ou uma seção mencionando normas aplicadas no desenvolvimento e uso de Caches;

3. Não existe nenhum tipo de modelo para a análise de Cache de dispositivos móveis. O que foi encontrado foram modelos de análise para dispositivos móveis, como descrito na Seção 2.3 no Capítulo 2, mas não um modelo específico para análise de Cache de dispositivos móveis.

Diante disso, pode-se ressaltar que o objetivo do trabalho foi atendido. Para alcançar esse objetivo foi proposto um modelo de análise de Cache em dispositivos móveis. Outro ponto foi a definição de fatores de avaliação, o que permitiu uma avaliação das bibliotecas e APIs que implementam Cache e ainda a implementação do protótipo funcional que auxiliou na avaliação das bibliotecas com base nos fatores definidos.

6.2 Dificuldades Encontradas

No que tange a dificuldades para realizar este trabalho, a principal delas se refere a diversidade de formatos de arquivos de Cache, sem uma estrutura de arquivo conhecida e sem documentação específica. Outra dificuldade foi no desenvolvimento de um protótipo que pudesse englobar todas as bibliotecas, o que não foi possível pelo tempo, pois determinadas bibliotecas não possuem qualquer tipo de suporte documental, perdendo-se muito tempo para implementação e testes.

6.3 Trabalhos Futuros

Como trabalhos futuros, pode-se:

1. As bibliotecas analisadas envolveram apenas as que trabalham com dados salvos em disco. Há necessidade que se avalie os dados também em memória;

2. Analisar uma lista de aplicativos da Google PlayStore, identificando quais bibliotecas de Cache são implementadas por tais aplicações, fazendo uma lista ranqueada sobre os aplicativos, baseado nas deficiências encontradas nesta dissertação sobre cada biblioteca de Cache, averiguando se existe a necessidade de correção em detrimento aos apontamentos que foram feitos, relacionados à segurança permanecem;

3. Criar um mecanismo de análise capaz de averiguar aplicações que fazem uso de bibliotecas de Cache;

4. Incrementar tal mecanismo de análise no sentido de classificar que tipos de dados estão sendo salvos no dispositivo conforme uma categoria de privacidade pré-estabelecida.

5. Analisar que tipo de Criptografia as bibliotecas de cache (as que empregam) estão utilizando, buscando evidenciar se dados sensíveis do usuário estão empregando ou não tais tipos de criptografia/ofuscação.

Referências Bibliográficas

[1] S. Finley and X. Du, "Dynamic cache cleaning on android," in *Communications (ICC), 2013 IEEE International Conference on*, pp. 6143–6147, IEEE, 2013.

[2] S.-E. Kim, J.-T. Jo, and S.-H. Choi, "Sms spam filterinig using keyword frequency ratio," *SERSC: International Journal of Security and Its Applications*, vol. 9, no. 1, pp. 329–336, 2015. http://dx.doi.org/10.14257/ijsia.2015.9.1.31.

[3] M. S. Vani, R. Bhramaramba, D. Vasumati, and O. Y. Babu, "Tui based touch-spam detection in mobile applications to increase the security from advertisement networks," *IJACCC: International Journal of Advanced Computer Communications and Control*, vol. 02, no. 01, pp. 17–22, 2014. http://goo.gl/S1sKSx.

[4] S. Kim and H. Jin, "A simple security architecture for mobile office," *SERSC: International Journal of Security and Its Applications*, vol. 9, no. 1, pp. 139–146, 2015. http://dx.doi.org/10.14257/ijsia.2015.9.1.15.

[5] U. Mardikar, K. Griffin, E. Miller, and A. Patel, "Mobile anti-phishing," Feb. 18 2014. US Patent 8,656,459.

[6] N. Virvilis, N. Tsalis, A. Mylonas, and D. Gritzalis, "Mobile devices: A phisher's paradise," in *Proc. of the 11th International Conference on Security and Cryptography*, pp. 79–87, 2014.

[7] A. J. Alzahrani and A. A. Ghorbani, "Sms mobile botnet detection using a multi-agent system: Research in progress," in *Proceedings of the 1st International Workshop on Agents and CyberSecurity*, ACySE '14, (New York, NY, USA), pp. 2:1–2:8, ACM, 2014.

[8] S. Pu, Z. Chen, C. Huang, Y. Liu, and B. Zen, "Threat analysis of smart mobile device," in *General Assembly and Scientific Symposium (URSI GASS), 2014 XXXIth URSI*, pp. 1–3, IEEE, 2014.

[9] Android, "Security tips," 2014. `https://developer.android.com/training/articles/security-tips.html`.

[10] T.-E. Wei, A. B. Jeng, H.-M. Lee, C.-H. Chen, and C.-W. Tien, "Android privacy," in *Machine Learning and Cybernetics (ICMLC), 2012 International Conference on*, vol. 5, pp. 1830–1837, IEEE, 2012.

[11] A. Shabtai, Y. Elovici, and L. Rokach, *A survey of data leakage detection and prevention solutions*. Springer Science & Business Media, 2012.

[12] VIAFORENSICS, "White paper:appwatchdog findings - sensitive user data stored on android and iphone devices," 2011. `http://viaforensics.com/appwatchdog`.

[13] Android, "Storage options - saving cache files," 2014. `http://developer.android.com/guide/topics/data/data-storage.html#filesIntern`.

[14] Symantec, "Istr20 - internet security threat report," 2015. `https://goo.gl/InAuxE`.

[15] Symantec, "Pesquisa norton mobile," 2014. `http://goo.gl/pWSL2E`.

[16] A. M. Braga, E. N. do Nascimento, and L. Rodrigues, "Introdução à segurança de dispositivos móveis modernos–um estudo de caso em android," *Simpósio em Segurança da Informação e de Sistemas Computacionais. Sociedade Brasileira de Computação–SBC*, vol. 12, 2012.

[17] GARTNER, "Gartner says worldwide smartphone sales recorded slowest growth rate since 2013," 2015. `http://www.gartner.com/newsroom/id/3115517`.

[18] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale," in *Trust and Trustworthy Computing* (S. Katzenbeisser, E. Weippl, L. Camp, M. Volkamer, M. Reiter, and X. Zhang, eds.), vol. 7344 of *Lecture Notes in Computer Science*, pp. 291–307, Springer Berlin Heidelberg, 2012.

[19] D. Cibrão and R. Gonçalves, "Segurança no android," 2015. `http://goo.gl/Wh5PHx`.

[20] Guy Podjarny, "Understanding mobile cache sizes," 2011. `http://www.guypo.com/understanding-mobile-cache-sizes/`.

[21] A. Hoog, *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*. Elsevier, Inc., 2011.

[22] TECHOTOPIA, "An overview of the android architecture," 2014. www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture.

[23] C.-W. Chang, C.-Y. Lin, C.-T. King, Y.-F. Chung, and S.-Y. Tseng, "Implementation of jvm tool interface on dalvik virtual machine," in *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*, pp. 143–146, April 2010.

[24] C. Enrique Ortiz, "Entendendo seguranÃ§a no android," 2010. http://www.ibm.com/developerworks/br/library/x-androidsecurity/.

[25] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, (New York, NY, USA), pp. 627–638, ACM, 2011.

[26] Android, "Lrucache," 2015. http://developer.android.com/reference/android/util/LruCache.html.

[27] V. Brison, "Readme: Android dualcache," 2014. https://github.com/vincentbrison/android-easy-cache.

[28] Android, "Copyright (c) 2011 the android open source project," 2011. https://android.googlesource.com/platform/libcore/+/android-4.1.1_r1/luni/src/main/java/libcore/io/DiskLruCache.java.

[29] Fhucho, "simple-disk-cache," 2015. https://github.com/fhucho/simple-disk-cache.

[30] C. Andrews, "Readme: Httpresponsecache," 2012. https://github.com/candrews/HttpResponseCache.

[31] N. Jafelle, "Qachee," 2015. https://github.com/nicolasjafelle/Qachee.

[32] A. Cowkur, "Reservoir," 2015. https://github.com/anupcowkur/Reservoir.

[33] C. Banes, "Android-bitmapcache is a specialised cache, for use with android bitmap objects.," 2012. https://github.com/chrisbanes/Android-BitmapCache.

[34] I. Connor, "Objectcache," 2015. https://github.com/iainconnor/ObjectCache.

[35] Kinvey, "Ready to build amazing apps?," 2015. `http://devcenter.kinvey.com/android`.

[36] V. Rawat, "Expirabledisklrucache.," 2015. `https://github.com/vijayrawatsan/ExpirableDiskLruCache`.

[37] Facebook, "Introducing conceal: Efficient storage encryption for android," 2015. `https://goo.gl/fstxON`.

[38] Caceres, Evelio Tarazona, "Carbonite - a simple in memory and persistent object cache for android.," 2013. `https://github.com/eveliotc/carbonite`.

[39] Kryo, "Java serialization and cloning: fast, efficient, automatic," 2015. `https://github.com/EsotericSoftware/kryo`.

[40] A. E. Brill, M. Pollitt, and C. Morgan Whitcomb, "The evolution of computer forensic best practices: An update on programs and publications," *Journal of Digital Forensic Practice*, vol. 1, no. 1, pp. 3–11, 2006.

[41] S. R. Selamat, R. Yusof, and S. Sahib, "Mapping process of digital forensic investigation framework," *International Journal of Computer Science and Network Security*, vol. 8, no. 10, pp. 163–169, 2008.

[42] R. McKemmish, *What is forensic computing?* Australian Institute of Criminology, 1999.

[43] J. Slay, Y.-C. Lin, B. Turnbull, J. Beckett, and P. Lin, "Towards a formalization of digital forensics," in *Advances in Digital Forensics V* (G. Peterson and S. Shenoi, eds.), vol. 306 of *IFIP Advances in Information and Communication Technology*, pp. 37–47, Springer Berlin Heidelberg, 2009.

[44] G. M. Mohay, A. Anderson, B. Collie, R. D. McKemmish, and O. d. Vel, *Computer and intrusion forensics*. Artech House, 2003.

[45] G. Palmer *et al.*, "A road map for digital forensic research," in *First Digital Forensic Research Workshop, Utica, New York*, pp. 27–30, 2001.

[46] M. Reith, C. Carr, and G. Gunsch, "An examination of digital forensic models," *International Journal of Digital Evidence*, vol. 1, no. 3, pp. 1–12, 2002.

[47] S. Ó. Ciardhuáin, "An extended model of cybercrime investigations," *International Journal of Digital Evidence*, vol. 3, no. 1, pp. 1–22, 2004.

[48] K. Kent, S. Chevalier, T. Grance, and H. Dang, "Guide to integrating forensic techniques into incident response," *NIST Special Publication*, pp. 800–86, 2006.

[49] T. Vidas, C. Zhang, and N. Christin, "Toward a general collection methodology for android devices," *digital investigation*, vol. 8, pp. S14–S24, 2011.

[50] OWASP, "Static code analysis," 2016. `https://www.owasp.org/index.php/Static_Code_Analysis`.

[51] Z. Zhao and F. C. C. Osono, "Trustdroid: Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking," in *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*, pp. 135–143, IEEE, 2012.

[52] Z. Yang and M. Yang, "Leakminer: Detect information leakage on android with static taint analysis," in *Software Engineering (WCSE), 2012 Third World Congress on*, pp. 101–104, IEEE, 2012.

[53] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 1043–1054, ACM, 2013.

[54] M. Zhang and H. Yin, "Efficient, context-aware privacy leakage confinement for android applications without firmware modding," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp. 259–270, ACM, 2014.

[55] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[56] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, p. 29, ACM, 2014.

[57] J. McClurg, J. Friedman, and W. Ng, "Android privacy leak detection via dynamic taint analysis," 2013. `http://www.jrmcclurg.com/papers/internet_security_final_report.pdf`.

[58] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, R. Siegfried, E. Bodden, D. Octeau, and P. Mcdaniel, "IccTA: Detecting Inter-Component Privacy Leaks in Android Apps," in *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*, 2015.

[59] P. Lam, E. Bodden, O. Lhoták, and L. Hendren, "The soot framework for java program analysis: a retrospective," in *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*, 2011.

[60] K. Yang, J. Zhuge, Y. Wang, L. Zhou, and H. Duan, "Intentfuzzer: detecting capability leaks of android applications," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp. 531–536, ACM, 2014.

[61] A. Takanen, J. D. Demott, and C. Miller, *Fuzzing for software security testing and quality assurance*. Artech House, 2008.

[62] C. Gibler, J. Crussell, J. Erickson, and H. Chen, *AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale*. Springer, 2012.

[63] S. Amini, J. Lindqvist, J. Hong, J. Lin, E. Toch, and N. Sadeh, "Caché: caching location-enhanced content to improve user privacy," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pp. 197–210, ACM, 2011.

[64] Y. Zhang, C. Tan, and L. Qun, "Cachekeeper: a system-wide web caching service for smartphones," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pp. 265–274, ACM, 2013.

[65] A. Anand, M. Reshadi, B. Du, H. Kolam, S. Jaiswal, and A. Akella, "A case for application-managed cache for browser," in *Multimedia and Expo (ICME), 2015 IEEE International Conference on*, pp. 1–6, IEEE, 2015.

[66] P. Stephenson, "Modeling of post-incident root cause analysis," *International Journal of Digital Evidence*, vol. 2, no. 2, pp. 1–16, 2003.

[67] Android, "The android open source project," 2010. `https://goo.gl/eUV5sN`.

[68] Android, "Disklrucache," 2011. `http://goo.gl/rPjw5f`.

[69] OWASP, "Owasp mobile security project," 2015. `https://goo.gl/cXOJhD`.

[70] WinkToolKit, "Html5 application cache study," 2015. `http://www.winktoolkit.org/blog/235`.

[71] M. J. Dworkin, "Sp 800-38a addendum. recommendation for block cipher modes of operation: Three variants of ciphertext stealing for cbc mode," tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, United States, 2010.

[72] ZETETIC, "Full database encryption for sqlite," 2015. `https://www.zetetic.net/sqlcipher/`.

[73] ZETETIC, "Design, security features," 2015. `https://www.zetetic.net/sqlcipher/design/`.

[74] Android, "Security tips," 2015. `http://goo.gl/kb4ReQ`.

[75] Holanda, Maristela Terto de and Fernandes, Jorge Henrique Cabral, "SeguranÃ§a no desenvolvimento de aplicações," 2011. `http://goo.gl/jbwhx4`.

Apêndice A

Armazenamento de Figuras em Cache

A Tabela A.1 exemplifica como as bibliotecas para Cache de imagem salvam as figuras utilizadas na avaliação.

Tabela A.1: Lista de Figuras

Nome do arquivo em cache	Imagem no servidor
0eb4da80b5d28bfd34666355986c2b9c.0	http://29.media.tumblr.com/tumblr_lht5uy6khS1qed3e3o1_500.jpg"
1d48739dbeae4766729203a5f2267d5f.0	http://24.media.tumblr.com/tumblr_ltzh55Xi7J1r1vf30o1_500.jpg
1e0ef22d1c8d6de0a29c4e93c8d9fe45.0	http://24.media.tumblr.com/tumblr_lsvcpxVBgd1qzgqodo1_500.jpg
1ed53565eaaa0e49c867dd2573fa7434.0	http://27.media.tumblr.com/tumblr_lqf1sn86te1qaa50yo1_500.jpg
2ca26ee914bb7ea13c67bd0f63bdb2ff.0	http://24.media.tumblr.com/tumblr_lsvcpxVBgd1qzgqodo1_500.jpg
2ecc0698d12e2fb9733e140bf230164d.0	http://26.media.tumblr.com/tumblr_ljju0frmo81qaa50yo1_400.jpg
2ecd17f841dafa90c9c807059a129597.0	http://28.media.tumblr.com/tumblr_lja6xpfr4X1qaa50yo1_500.jpg
3b6aa487c1b1f69e0c20a31c883cfa36.0	http://29.media.tumblr.com/tumblr_lu7ep6O5hr1qbjvpuo1_500.jpg\
4b815426962010baa0cd33aaf3b3d547.0	http://ilovepugs.tumblr.com/post/12403103214/pugs-kylielovestrees-img-7905.jpg
5ea6a27eec4e4a50bf6f586a3c7d958e.0	http://24.media.tumblr.com/tumblr_mc00rmFhJI1rh08hdo1_500.jpg
05f52c13ddc2de0eb5e0b5d9596d2f54.0	http://41.media.tumblr.com/tumblr_mavsumGGAd1qboaw8o1_500.jpg
006e9507e0e56ded0c7d9594497d2215.0	http://27.media.tumblr.com/tumblr_ltuo57ahqE1qa6z3eo1_500.jpg
7ad24e4c4501a36f1b68be625d0777df.0	http://29.media.tumblr.com/tumblr_lsx6cf3Wkg1qb08qmo1_500.jpg
7cc556b40dab9867f0c8c77dca2867e5.0	http://26.media.tumblr.com/tumblr_lil8a1m1YM1qzj3syo1_500.jpg
8e92c60ad398b387872e88a7e8706ae7.0	http://26.media.tumblr.com/tumblr_lhkn2zWJdn1qzgqodo1_400.jpg
9aeaca51c461abe9eced629fb3ed3b2c.0	http://26.media.tumblr.com/tumblr_loiqhpYqqQ1qaa50yo1_500.jpg
9c9aa5edf1e41425c35bdb32acf9fb54.0	http://29.media.tumblr.com/tumblr_lsvczkC8e01qzgqodo1_500.jpg
9e0c38eac230caa89bf53efcc0e25e16.0	http://26.media.tumblr.com/tumblr_lttbk92Ko01ql9nqgo1_500.jpg
10f25d52f5618a2174c23dfa4a1e6127.0	http://27.media.tumblr.com/tumblr_ll3xua50Vr1qb08qmo1_500.jpg
9e124a2627736b88e36f2403eac58a7d.0	http://24.media.tumblr.com/tumblr_mblnsnvibw1r3pfomo1_500.jpg
24c52ace61c49ec2ccf4f218a608dee9.0	http://31.media.tumblr.com/tumblr_mc4u6lwZHr1qf4k86o1_500.jpg
25eeb2297751c5925cf9898e7769c94d.0	http://27.media.tumblr.com/tumblr_liy1xfY9G71qftdfxo1_500.jpg
44d528c8158dea9c8ef3f78f8d38c15c.0	http://26.media.tumblr.com/tumblr_lrqnevtBvM1qb08qmo1_400.jpg
58c07527e40ac57074113f07ad090ade.0	http://24.media.tumblr.com/tumblr_mbncuaqA661qzio10o1_400.jpg
61e48c929f6010e9dba3954a3ea01bdf.0	http://33.media.tumblr.com/tumblr_mbnbhvErFJ1qb08qmo1_500.jpg
67e89b733fced4785cf26715de6a9810.0	http://27.media.tumblr.com/tumblr_ll2253YjYU1qjkqano1_500.jpg
76e4ae5a03ece10fb6312761d3bcc878.0	http://27.media.tumblr.com/tumblr_lsvlcmTElk1r1z2mqo1_500.jpg
77cb40eca3adc35d8469cef6050cade3.0	http://28.media.tumblr.com/tumblr_locinzasB91qzj3syo1_500.jpg
94d709f55792eabd68c5a2205f0981fb.0	http://24.media.tumblr.com/2e5eb3171fc933f19786610345879fcf/tumblr_mozdou8yEC1qb08qmo1_500.jpg
100a2dbc004b4bda71d92af95e0ae540.0	http://28.media.tumblr.com/tumblr_lj0eomAZZ91qb08qmo1_500.jpg
105a01a3f1b71d96aa9c3385ccdf836f.0	http://24.media.tumblr.com/tumblr_lsvczkC8e01qzgqodo1_500.jpg
216bf745167d08fd2e684323d02658c1.0	http://41.media.tumblr.com/2e5eb3171fc933f19786610345879fcf/tumblr_mozdou8yEC1qb08qmo1_500.jpg
302cece025225d7505fd7d99e0d84fde.0	http://30.media.tumblr.com/tumblr_lj53czBbvd1qzgqodo1_500.jpg
581d35426ed30f44d231919d5c9b5170.0	http://24.media.tumblr.com/tumblr_mbs9uw4Uoy1qaa50yo1_500.jpg
639a940785ba5e34f52559f844ad75dd.0	http://28.media.tumblr.com/tumblr_ls9fa8S9lE1qfbmwho1_500.jpg
815bd878cb4444020e98b64c1a5137be.0	http://30.media.tumblr.com/tumblr_ltslj7ykIi1qlhdjjo1_500.jpg
924e69220afbbd3bb43257c190ded60c.0	http://25.media.tumblr.com/tumblr_li5uwk1P0W1qepvs6o1_400.jpg
4568ed669211a7fb7ec1c675f7d2c321.0	http://26.media.tumblr.com/tumblr_lji145OZao1qaa50yo1_500.jpg
80380c8f92f95a0d846dea324a8596fd.0	http://27.media.tumblr.com/tumblr_lhq4onW6Vg1qbcihro1_500.jpg
503104ef5a6d03f1852b09b167829443.0	http://29.media.tumblr.com/tumblr_lisw5rptyA1qbbpjfo1_500.jpg
60659657f8d51e6d2c70d68d025d0fa1.0	http://26.media.tumblr.com/tumblr_lomvroWFOE1qaa50yo1_500.jpg
a1e50329954b81791f70d21b7ae50e9c.0	http://28.media.tumblr.com/tumblr_lieud1GU541qaa50yo1_500.jpg
a22a80405091cd1b4e4a8b0bae6f0f60.0	http://38.media.tumblr.com/tumblr_mcgpapmE3I1qkoat7o1_400.jpg
a47ddeb9046ddccddff9fc5d3b2755e0.0	http://30.media.tumblr.com/tumblr_liyjwy0bKP1qenyvto1_500.jpg
c6a8566753f105dea4917a23377321d1.0	http://41.media.tumblr.com/2e5eb3171fc933f19786610345879fcf/tumblr_mozdou8yEC1qb08qmo1_500.jpg
c97ae785cd23d0802430e5c444bd4950.0	http://38.media.tumblr.com/tumblr_maxm4ecVNq1qjmqavo1_500.jpg
c981e6db1a9b628900323282a359531b.0	http://40.media.tumblr.com/2e5eb3171fc933f19786610345879fcf/tumblr_mozdou8yEC1qb08qmo1_500.jpg
cc3f992e4efb3b0da8be5446fad20d61.0	http://25.media.tumblr.com/tumblr_mbnbhvErFJ1qb08qmo1_500.jpg
cce0485eab376c096ddbbb7b303f4d1b.0	http://38.media.tumblr.com/tumblr_mbnh52cctT1r3ip8io1_500.jpg
cf358c405acb8419dc2f0258dd9e3165.0	http://24.media.tumblr.com/tumblr_lixgdemCvf1qaa50yo1_500.jpg
d3fd551db363497eaeb10a2844c72c3d.0	http://30.media.tumblr.com/tumblr_lsvlgkoGXf1r1z2mqo1_500.jpg
d8b87682d4007913246f81eb92452b57.0	http://31.media.tumblr.com/95a84579fa297844891b3ab1a5c76c0a/tumblr_mooibbcKl51rylzllo1_500.jpg
d8cfa278176746b07d6b96a0398b4086.0	http://29.media.tumblr.com/tumblr_lifvawnQjo1qaa50yo1_500.jpg
d9d1a70dece2728a62b90be46500bdab.0	http://30.media.tumblr.com/tumblr_lsx4nsg5Hj1qa25vco9_500.jpg
d10f2675f393c74abffa91db49e6e112.0	http://27.media.tumblr.com/tumblr_lj4x7uXySA1qcbrufo1_500.jpg
d061b9ac302e679fe9a351bf6f954f2c.0	http://26.media.tumblr.com/tumblr_lsigg3D23h1qz9wudo1_500.jpg
d61825722f4d8bd11abf2d6841467a1d.0	http://24.media.tumblr.com/tumblr_lu7ep6O5hr1qbjvpuo1_500.jpg
db80c913022c9c0bd07e34c6b4fbddb7.0	http://ilovepugs.tumblr.com/post/12403103214/pugs-kylielovestrees-img-7905.jpg
ddd45e5df5b6a37d90bfaad61eaae3d2.0	http://41.media.tumblr.com/tumblr_mc4u6lwZHr1qf4k86o1_500.jpg
df914bf926c50bc0aa3c9f56945e12db.0	http://25.media.tumblr.com/tumblr_lttbk92Ko01ql9nqgo1_500.jpg
dfbcc25d3585c9b28b3da13aac1e5d6c.0	http://29.media.tumblr.com/tumblr_li4ww5mDMf1qb08qmo1_500.jpg
ec84a7c17e9cc96056db3dc6dbc8faae.0	http://26.media.tumblr.com/tumblr_lrqnevtBvM1qb08qmo1_400.jpg