
**UNIVERSIDADE FEDERAL DE PERNAMBUCO**
**CENTRO DE INFORMÁTICA**

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO PAULO TENÓRIO TRINDADE

# IncR: FERRAMENTA DE DETECÇÃO INCREMENTAL DE COMUNICAÇÃO ENTRE COMPONENTES ANDROID

RECIFE
2016

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

JOÃO PAULO TENÓRIO TRINDADE

# IncR: FERRAMENTA DE DETECÇÃO INCREMENTAL DE COMUNICAÇÃO ENTRE COMPONENTES ANDROID

Trabalho de graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Leopoldo Motta Teixeira

RECIFE
2016

**JOÃO PAULO TENÓRIO TRINDADE**

IncR: FERRAMENTA DE DETECÇÃO INCREMENTAL DE COMUNICAÇÃO ENTRE COMPONENTES ANDROID

Trabalho de graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

APRESENTAÇÃO em
Recife, _____ de _____ de 2016.

BANCA EXAMINADORA
Orientador: Leopoldo Motta Teixeira

_____

Avaliador: Henrique Emanuel Mostaert Rebêlo

_____

Recife
2016

## AGRADECIMENTOS

Sobre tudo e sobre todos agradeço a Deus, pois é nEle que eu encontro o sentido da vida.

Dedico este trabalho à Sílvia Tenório, minha amada mãe, que dedica toda a sua vida para o meu sucesso e felicidade. Sempre me deu o suporte mais que necessário para que me empenhasse exclusivamente a estudar e me preparar para a vida. Sem o seu suporte tudo teria sido mais difícil.

Dedico também à Aluísio Aldo, que considero mais que um pai para mim. Sempre me inspirei no seu exemplo de superação e sucesso. Jamais terei como retribuir tudo que você me proporcionou. A você, minha eterna gratidão.

Agradeço a minha tia Rosa que sempre me alertou para a importância da educação e incondicionalmente sempre me apoiou quando precisei.

Agradeço àquela que esteve ao meu lado durante todos esses anos de graduação, Karine Maia. Além de namorada, você é minha parceira em tudo. Saiba que sonhar com um futuro ao seu lado sempre foi uma motivação para continuar e chegar até aqui. Muito obrigado pelo seu carinho e amor.

Aos meus queridos amigos de estágio e faculdade, Leu e Miguel, meu muito obrigado pela parceria e pelo amadurecimento acadêmico e profissional que o convívio diário com vocês me proporcionou. Aprendi bastante coisa com vocês.

Aos meus irmãos de Pequeno Grupo, o PG da zona oeste, muito obrigado pelas orações e pelo apoio espiritual. É muito bom estar com vocês todas as terças.

Aos meus queridos irmãos do IBCIn Renata, Joelma, Edna, Katia, Vanessa, Wagner e Yeda, muito obrigado pelas orações e pelo carinho de todos vocês. Deus saberá retribuir a dedicação de cada um por realizarem uma obra tão maravilhosa. Grandes coisas Deus fez e continuará fazendo através da vida de vocês.

Agradeço a todos os professores do Centro de Informática da UFPE pela educação de excelência que vocês ajudam a construir diariamente, em meio a todos os desafios que é fazer educação no Brasil.

Agradeço ao meu professor orientador, Leopoldo Teixeira, pela disponibilidade, paciência e o quanto me ajudou a concluir este trabalho.

Termino agradecendo mais uma vez Àquele que é o início e fim. Dedico a Ele tudo o que tenho e o que sou. Minha eterna gratidão pela Sua imensa misericórdia e maravilhosa graça.

“Pensava que nós seguíamos caminhos já feitos, mas parece que não os há. O nosso ir faz o caminho. ”

(C.S. Lewis)

## RESUMO

A plataforma Android é a mais popular no mercado de dispositivos móveis. E isso se dá, entre outros fatores, graças a enorme quantidade de opções de aplicativos disponíveis para seus usuários. Uma das principais características de suas aplicações é a capacidade de comunicação entre elas, e isso acaba criando um potencial canal de vazamento de dados sensíveis dos usuários. Além disso, as ferramentas criadas para detectar esses canais vulneráveis apresentam certas limitações de escalabilidade e quando se considera a análise de um contexto volátil de aplicativos, com sucessivas instalações, desinstalações e atualizações.

Este trabalho propõe e analisa a ferramenta IncR, capaz de identificar incrementalmente canais de comunicação entre aplicativos Android. Nesse contexto, incremental significa combinar os resultados já obtidos em execuções anteriores sem precisar refazer as análises com resultados já conhecidos, considerando a volatilidade de contexto citada. Os resultados das análises do IncR demonstram que sua abordagem escala de forma bem mais eficiente quando comparada com a abordagem padrão.

## ABSTRACT

The Android platform is currently the most popular in the mobile marketplace, due to the large variety of applications available to users, among other factors. Communication between applications represents one of the main features of Android Applications, but this can lead to user sensitive data leaks. Moreover, existing tools that try to identify these vulnerability points have scalability limitations. Moreover, we also need to consider the mutable state of current installed applications, with successives installations, uninstallations and updates on the apps.

This work proposes and evaluates the IncR tool, capable of incrementally tracking communications channels between Android applications. In this context, incremental means combining obtained results from previous executions without the need to redo the same evaluations already performed in the known results, but considering the mutable state, as mentioned. The results of the IncR evaluation show that the incremental approach scales more efficiently when compared with the traditional approach.

# LISTA DE FIGURAS

# LISTA DE TABELAS

## SUMÁRIO

# 1. INTRODUÇÃO

Android é uma das plataformas móveis mais populares do mundo. Domina a maior parte do mercado móvel, e além de ser mantida por uma das maiores empresas de tecnologia, a Google, um dos fatores que mais contribuem para sua popularidade está relacionado a sua enorme variedade de aplicações desenvolvidas e disponíveis para *download* em sua loja de aplicativos.

Com toda essa popularidade e considerando o gigantesco número de sua base de usuários ativos, 1.4 bilhões em Setembro de 2015 [Android Central, 2015], alguns estudos quanto a vulnerabilidade da plataforma se fazem necessários para mitigar os riscos relativos aos dados desses usuários. Um cenário muito comum no Android é a comunicação entre aplicativos, que acontece por meio de envio de mensagens. Essas mensagens podem conter dados do usuário, inclusive com informações sensíveis, que se forem interceptadas por algum outro aplicativo mal-intencionado podem causar sérios problemas.

A própria plataforma Android inclui um mecanismo capaz de identificar comunicação entre aplicativos, entretanto, isso só é possível em tempo de execução, o que deixa bastante caro, do ponto de vista de processamento, reproduzir esse mecanismo para testes de vulnerabilidade no contexto de um dispositivo com centenas de aplicativos instalados. Além disso, é interessante analisar múltiplos contextos, uma vez que usuários podem instalar aplicativos de forma arbitrária em seus dispositivos. Por causa disso, surgiram algumas ferramentas capazes de executar análises em tempo de compilação, ou seja, a partir do código-fonte das aplicações. Entretanto, essas ferramentas possuem limitações quanto a escalabilidade, e quanto a análise de contexto volátil, com sucessivas instalações, desinstalações e atualizações de aplicativos.

## 1.2 OBJETIVOS

O objetivo geral deste trabalho é desenvolver um ambiente que seja capaz de identificar de forma incremental a comunicação entre aplicações na plataforma Android. Para atingir o objetivo geral, alguns objetivos específicos foram traçados:

I. Projetar um algoritmo que descreve a detecção incremental da comunicação entre aplicativos Android.
II. Desenvolver uma ferramenta que implementa o algoritmo projetado.
III. Avaliar o desempenho da ferramenta desenvolvida.

## 1.3 ESTRUTURA DO DOCUMENTO

O restante deste documento está organizado da seguinte maneira: o Capítulo 2 apresenta uma revisão bibliográfica com conceitos importantes para o entendimento geral deste trabalho, tais como conceitos básicos do Android, comunicação entre componentes e vazamento de dados. O Capítulo 3 descreve os objetos desenvolvidos neste trabalho: o algoritmo *Incremental Resolution* e a ferramenta IncR. O Capitulo 4 descreve uma série de experimentos realizados com a ferramenta IncR e analisa os resultados obtidos. Por fim, o Capítulo 5 faz um apanhado geral, apresentando conclusões a partir do que foi construído neste trabalho, e sugestões de possíveis trabalhos futuros.

# 2. REVISÃO BIBLIOGRÁFICA

Neste capítulo, apresentamos conceitos fundamentais acerca da plataforma Android, tais como arquitetura, componentes básicos e a comunicação entre eles através do mecanismo de *Intents*. Em seguida, apresentamos como as aplicações podem se comunicar, e qual o papel do mecanismo de *Intent Resolution* nesse processo. Por fim, discutimos sobre a possibilidade de vazamento de dados como efeito da comunicação entre aplicativos.

## 2.1. ANDROID

Android é um sistema operacional baseado no núcleo Linux [Wikipédia, 2016a], inicialmente projetado para *smartphones*, e que rapidamente alcançou toda uma gama de dispositivos móveis, como *tablets* e *smartwatches* (Android Wear). Também existem versões projetadas para *Smart* TVs (Android TV) e carros (Android Auto).

Atualmente é a plataforma móvel mais popular no mundo, atingindo 80,7% das vendas de dispositivos móveis no último trimestre de 2015, segundo pesquisa realizada pela Gartner [Gartner, 2016]. Além disso, sua loja de aplicativos atingiu a expressiva marca dos 2 milhões de aplicativos disponíveis para serem baixados, no início de 2016 [Statista, 2016]. No momento da escrita deste trabalho, encontra-se na versão 6.0 Marshmallow e muito próximo de lançar a versão 7.0 Nougat.

### 2.1.1. Arquitetura

A arquitetura do sistema Android é composta pelas seguintes camadas, como ilustra a Figura 1:

- ***Kernel (Linux Kernel)*:** Uma versão do *kernel* do Linux modificada para melhor atender às limitações dos dispositivos móveis [Bordin, 2012]. É base da arquitetura onde encontram-se os programas de gerenciamento de energia e de memória, configurações de segurança, os *drivers* de hardware, e introduz o *Binder IPC driver*, um mecanismo que permite comunicação entre processos, ou *inter-process communication* (IPC) [Bordin, 2012];

- ***Hadware Abstraction Layer (HAL)*:** Interface que os fornecedores de hardware precisam implementar. Para Android, trata-se de uma abstração das implementações de *drivers* de baixo nível, permitindo o desenvolvimento de funcionalidades sem afetar as camadas acima [Android Source Project, 2016];
- ***Android Runtime*:** Contém um conjunto de bibliotecas do núcleo Java (*Core Libraries*), a *Dalvik Virtual Machine* (DVM) e o *Android RunTime* (ART). ART e Dalvik são *runtimes* usadas pelos aplicativos e alguns serviços de sistema no Android, e que operam *Dex bytecode*, garantindo, em certo nível, que sejam compatíveis. Em geral, aplicativos desenvolvidos para Dalvik também funcionam quando executam com ART. O ART é sucessor do Dalvik e está disponível a partir da versão 4.4 trazendo alguns recursos mais sofisticados, como por exemplo, melhorias no *Garbage Colletcion* com redução no uso de memória e fragmentação, e compilação *Ahead-of-time*, o que trouxe um crescimento significativo de performance nos aplicativos [Android Source Project, 2016];
- ***Native Libraries*:** Camada que dispõe de um conjunto de bibliotecas que são usadas pelo sistema para lidar com diferentes tipos de dados. Por exemplo, bibliotecas para suporte a formatos de áudio, vídeo e imagens, bibliotecas para gráficos 2D e 3D, e o próprio banco de dados SQLite [Bordin, 2012];
- ***Application Framework*:** Contém ferramentas responsáveis por gerenciar as funções básicas do dispositivo. Fornecem vários serviços aos desenvolvedores. O *Activity Manager,* por exemplo, é responsável por gerenciar o ciclo de vida das *Activities*, bem como organizá-las em pilhas e *tasks.* Outro exemplo é o *Notification Manager que permite* que aplicativos disparem notificações para o usuário [Bordin, 2012];
- ***Applications*:** Camada mais alta da arquitetura, caracteriza-se por prover a interação entre o dispositivo e o usuário. É onde estão os aplicativos instalados.

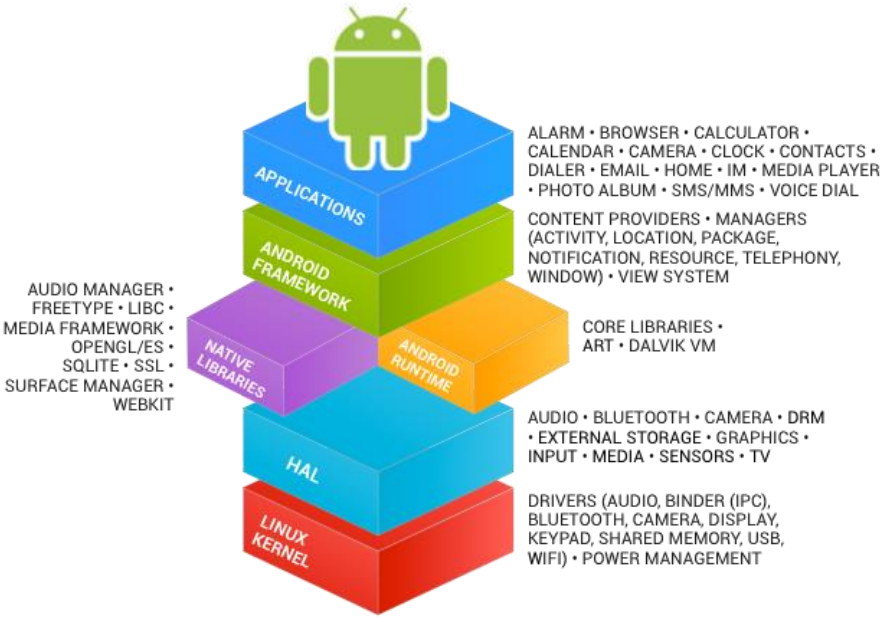


*Figura 1 - Camadas da arquitetura do Android [Android Source Project, 2016]*

### 2.1.2. Componentes

Os aplicativos desenvolvidos para Android são escritos na linguagem Java e são compostos por blocos básicos chamados de componentes [Android Developers, 2016a]. São eles:

1. *Activity:* corresponde a uma tela do aplicativo com uma interface do usuário que é independente e que pode operar em conjunto com outras telas. Por exemplo, uma tela de *login* é implementada como uma *Activity*;
2. *Service:* componente responsável por operações executadas em segundo plano, isto é, que não exigem interface com usuário. O seu uso é indicado em operações para processos remotos e de longa duração. Por exemplo, um *service* pode ser usado para implementar um tocador de música que continua executando mesmo quando o usuário sai do aplicativo;
3. *Content Provider:* é responsável pelo gerenciamento de um conjunto de dados compartilhados. É possível que outros aplicativos com permissão tenham acesso e até modifiquem os dados. Por exemplo, outras aplicações podem usar o *Content Provider* para acessar e até editar a lista de contatos no telefone com a permissão adequada;
4. *Broadcast Receiver*: componente sem interface com usuário e que desempenha o papel de receptor de mensagens enviadas em *broadcast* de

origem do próprio sistema, como por exemplo o aviso de conexão com a internet estabelecida, ou iniciadas por outros aplicativos.

Essa estrutura com base em componentes provê flexibilidade, reuso, compartilhamento de dados e funcionalidades, possibilitando a comunicação entre eles e, por fim, entre aplicações. A comunicação entre *Activities, Services e Broadcast Receivers* de aplicações diferentes, ou não, acontece por meio do envio e recebimento de mensagens assíncronas [Android Developers, 2016b], chamadas de *Intents*.

### 2.1.3. Intents

As *Intents* representam, conceitualmente, a intenção de execução de uma determinada ação e carregam consigo o conteúdo necessários para efetivar o envio e/ou recebimento de informações. São destinadas a ativar outros componentes e isso pode acontecer de forma explícita ou implícita.

Uma *Intent* explícita deve especificar qual componente será o recipiente da mensagem, isto é, ela declara o nome do componente alvo. Como pode ser visto na Figura 2, por exemplo, uma *Intent* que define "*DownloadService.class"* como seu componente alvo.

```
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

*Figura 2 - Exemplo de Intent Explícita.*

Por outro lado, uma *Intent* implícita não declara o nome do seu receptor, mas define uma ação específica a ser executada além de outras informações, permitindo que componentes desconhecidos a trate [Android Developers, 2016c]. Assim, é possível operar uma espécie de *late-runtime binding* (ou *late binding*), mecanismo este que permite utilizar um determinado componente em tempo de execução, sem que tenhamos previsto qual em tempo de compilação (DevMedia, 2007). Na Figura 3, por exemplo, a *Intent* define a ação como *"Intent.ACTION_SEND"* com objetivo de enviar uma mensagem de texto e sem se limitar a um aplicativo de email especifico, portanto ela não define nenhum componente alvo. Inclusive, é possível que qualquer

aplicativo de email se registre para receber *Intents* desse tipo, através do mecanismo exemplificado na Figura 6, o que será detalhado mais adiante.

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

*Figura 3 - Exemplo de Intent implícita*

## 2.2. INTER-COMPONENT COMMUNICATION (ICC)

Android dá suporte para comunicação entre aplicações, permitindo, entre outras coisas, o reuso de funcionalidades.

Por exemplo, um aplicativo de edição de fotos não precisa implementar a funcionalidade de captura, bastando apenas enviar uma solicitação, por meio de uma *Intent*, ao aplicativo de câmera que execute aquela ação e responda enviando a nova foto tirada.

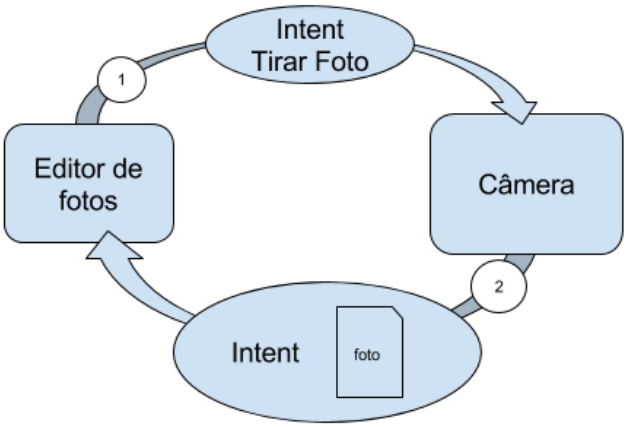


*Figura 4 - Esquema de solicitação e resposta entre dois aplicativos, via Intents.*

No Android isso é possível graças ao suporte dado a comunicação entre componentes, do inglês *Inter-Component Communication (ICC)*, que acontece através do envio e recebimento de *Intents*, também chamadas de *links ICC*.

Algumas ferramentas, como Epicc [Octeau et al, 2013] e IC3 [Octeau at al, 2015], foram desenvolvidas para detecção de *links* ICC. Entretanto, elas possuem limitações quanto a escalabilidade [Souza, 2016], e concentram-se apenas na detecção de pontos onde há o início da comunicação entre aplicativos, sem identificar os possíveis recipientes da mensagem. Nem sempre o fluxo de comunicação é definido explicitamente e, como já citado anteriormente, em vários casos acontece por meio do envio e recebimento de *Intents* implícitas, que são resolvidas apenas em tempo de execução.

Existe um mecanismo no Android que permite que um componente desconhecido seja testado e classificado como capaz ou não de corresponder a uma intenção de execução de uma determinada ação, representada por um *Intent* implícita, chamado de Resolução de Intenção, ou do inglês *Intent Resolution*.

## 2.3. INTENT RESOLUTION (IR)

Esse mecanismo opera todas as vezes que uma aplicação dispara para o sistema uma nova *Intent* implícita, de maneira que o sistema possa identificar quais aplicações instaladas naquele dispositivo possuem algum componente capaz de responder àquela *Intent* disparada.

Um elemento fundamental para a operação do *Intent Resolution* é o arquivo *AndroidManifest.xml* das aplicações receptoras, pois ele descreve para o sistema informações essenciais de um aplicativo Android [Android Developers, 2016d]. Nele estão informações de permissões, processos, a versão mínima do Android requerida, a lista de todos os componentes da aplicação, etc.

```
<activity
    android:name="com.github.pockethub.ui.MainActivity"
    android:theme="@style/Theme.PocketHub.NavigationDrawer"
    android:configChanges="orientation|keyboardHidden|screenSize"
    android:hardwareAccelerated="true">

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <meta-data
        android:name="android.app.default_searchable"
        android:value=".ui.search.SearchActivity" />
</activity>
<activity
    android:name="com.github.pockethub.ui.gist.CreateGistActivity"
    android:configChanges="orientation|keyboardHidden|screenSize"
    android:label="@string/create_gist">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/*" />
    </intent-filter>
</activity>
```

*Figura 5 - trecho de um arquivo AndroidManifest.xml.*

Para que o sistema execute o *Intent Resolution* é necessário que o componente alvo em potencial tenha discriminado no *AndroidManifest.xml* de seu aplicativo um recurso chamado de *Intent Filter*, ou filtro de intenções. Os *Intent Filters* possuem três principais elementos que especificam qual tipo de ação aquele componente é capaz de responder [Android Developers, 2016e], são eles:

- *action*: ação da *Intent*, como por exemplo, “android.intent.action.EDIT”;
- *data*: dados da *Intent* (URI e tipo de dados);
- *category*: categoria da *Intent*, como por exemplo, “android.intent.category.DEFAULT”;

```
<!-- This activity handles "SEND" actions with text data -->
<intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
</intent-filter>
```

*Figura 6 - Exemplo de Intent Filter e seus subelementos.*

Basicamente, o *Intent Resolution* compara a *Intent* disparada com os filtros de cada componente da aplicação alvo. Essa comparação é dividida em três operações [Android Developers, 2016f]:

1. **Teste de ação:** Teste responsável por comparar a ação definida pela *Intent* com lista de ações aceitas pelo componente alvo, isto é, as ações discriminadas em seu *Intent Filter*. Para que a ação da *Intent* seja aceita pelo componente, é preciso satisfazer pelo menos uma das duas condições:
    a. A ação da *Intent* corresponde a uma das ações definidas no filtro;
    b. A *Intent* não define nenhuma ação e o filtro contém pelo menos uma ação definida.
2. **Teste de categoria:** Teste responsável por verificar se o filtro do componente alvo contém todas as mesmas categorias definidas na *Intent*. Para que as categorias da *Intent* sejam aceitas pelo componente é necessário satisfazer uma das seguintes condições:
    a. Cada categoria definida na *Intent* possui uma categoria correspondente definida no filtro;
    b. A *Intent* não define nenhuma categoria.
3. **Teste de dados:** O teste de dados é responsável por comparar a *URI* e o tipo *MIME* definidos na *Intent* com os definidos no filtro do componente. Para que os dados da *Intent* sejam aceitos pelo componente é preciso satisfazer uma das seguintes condições:
    a. Nem a *Intent* nem o filtro possuem *URI* e *MIME*;
    b. A *Intent* e o filtro possuem *URI* com formatos correspondentes, e ambas não definem *MIME*;
    c. A *Intent* e o filtro possuem *MIME* correspondentes, e ambas não definem *URI*;
    d. A *Intent* e o filtro possuem *MIME* correspondentes, a *Intent* define uma *URI* de *'content:'* ou *'file:'* e o filtro não define *URI*;

e. A *Intent* e o filtro possuem *URI* e *MIME* correspondentes.

Uma vez que a *Intent* implícita disparada passa pelos três testes do *Intent Resolution*, aquele componente de outra aplicação está apto a responder à ação requisitada, possibilitando, em última análise, que aplicações distintas se comuniquem.

## 2.4. VAZAMENTO DE DADOS

Como as aplicações no Android utilizam o envio de mensagem como forma de comunicação, o fluxo de dados entre componentes, dada a troca de *Intents* entre si, emerge como potencial canal de vazamento de informações.

Isso acontece quando aplicativos maliciosos expõem intencionalmente alguma informação ou se aproveitam de um outro aplicativo “ingênuo” que envia informações sensíveis através de *Intents* implícitas. O vazamento de informações sensíveis em aplicações Android na comunicação entre componentes é também conhecida por *Component Hijacking* [Lu et al, 2012].

Algumas ferramentas, como as já citadas anteriormente Epicc e IC3, são capazes de detectar *links* ICC e que podem ser combinadas com outras ferramentas, como o IccTA [LI, 2014], que são capazes de detectar pontos de vulnerabilidade na comunicação entre componentes.

## 2.5. CONSIDERAÇÕES FINAIS

Este capítulo introduz conceitos básicos de Android como a arquitetura da plataforma, os componentes básicos de cada aplicativo e *Intents*. Além disso, apresenta os conceitos relacionados a comunicação entre aplicativos, tais como *Inter-Component Communication* e *Intent Resolution*. Também apresenta conceitos de riscos de vazamento de dados associados ao envio e recebimento de mensagens na *ICC*.

No próximo capítulo apresentamos alguns cenários envolvendo a identificação de ICC, bem como alguns problemas identificados. Por fim, apresentamos o algoritmo *Incremental Resolution* e a ferramenta IncR.

# 3. SOLUÇÃO

Neste capítulo apresentamos três cenários envolvendo detecção de ICC. Em seguida, dois problemas associados são discutidos, envolvendo a detecção não incremental de ICC. Por fim, apresentamos os dois principais resultados deste trabalho, o algoritmo *Incremental Resolution* e a ferramenta IncR.

## 3.1. CENÁRIOS

Seja $A_i$ um aplicativo Android, $C_j$ um conjunto conhecido de aplicativos Android e $R_k$ um subconjunto de $C_j$ formado somente por aqueles com os quais $A_i$ pode se comunicar.

### 3.1.1. Obtenção de $R_k$ a partir de $A_i$ e $C_j$

A partir de $A_i$ e $C_j$, deseja-se obter $R_k$. Para isso dois passos podem ser executados, como ilustra a Figura 7: primeiro, usar uma ferramenta, como Epicc ou IC3, para extração de *links ICC*, a partir do código-fonte de $A_i$. E por fim, de posse dos *links*, operar o *Intent Resolution* com os *Intent Filters* dos componentes dos aplicativos de $C_j$.

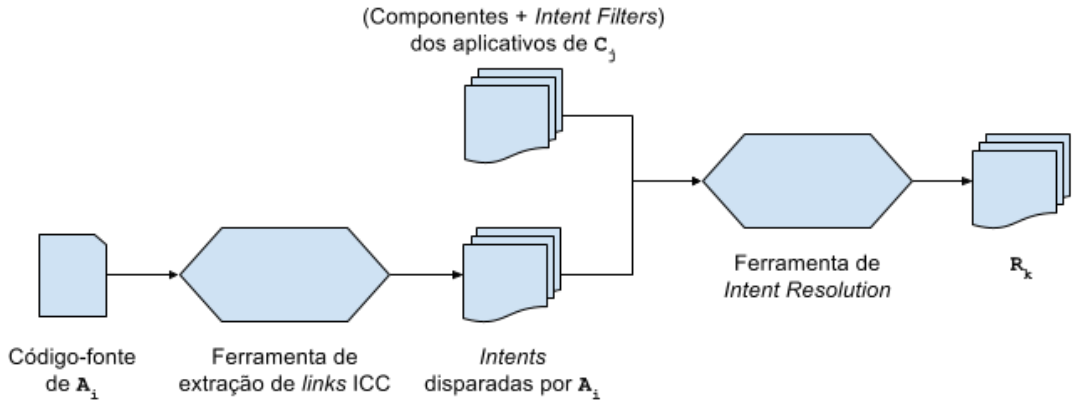


*Figura 7 - Obtenção de Intents de $A_i$ (à esquerda) e obtenção de $R_k$ (à direita).*

### 3.1.2. Obtenção de $R_{k+1}$ a partir de $A_i$ e $C_{j+1}$

A partir de $A_i$, $C_{j+1}$, $R_k$, onde $C_{j+1}$ foi obtido a partir da adição de um novo aplicativo $c$ em $C_j$, deseja-se obter $R_{k+1}$. Para isso, uma vez que o novo aplicativo $c$ já foi adicionado em $C_j$, os mesmos dois passos descritos no primeiro cenário podem ser executados, com as diferenças de que no passo 2 $C_{j+1}$ e $R_k$ são passados como entrada, como ilustra a Figura 8.

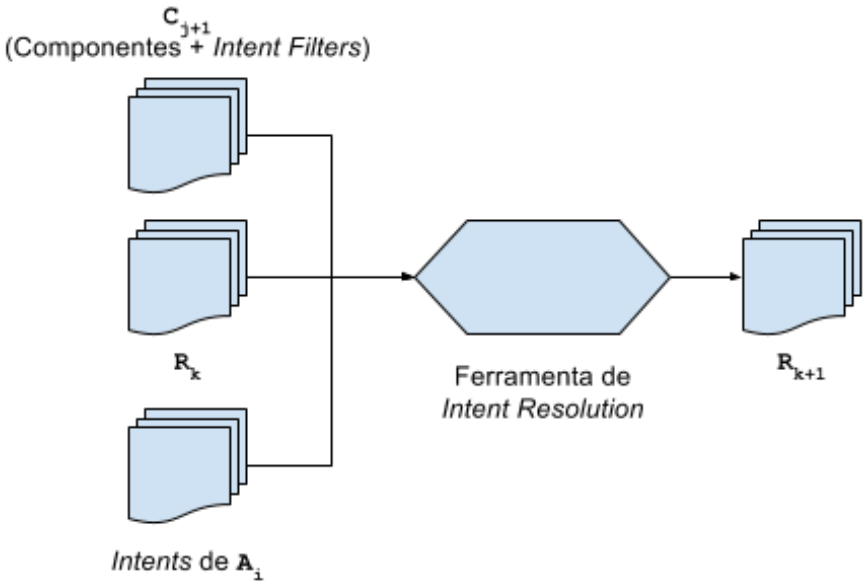


*Figura 8 - Obtenção de $R_{k+1}$ a partir $A_i$ e $C_{j+1}$.*

### 3.1.3. Obtenção de $R_{k+1}$ a partir de $A_{i+1}$ e $C_j$

A partir de $C_j$, $R_k$ e de $A_{i+1}$, onde $A_{i+1}$ corresponde a uma nova versão de $A_i$, deseja-se obter $R_{k+1}$. Também neste caso, os mesmos dois passos descritos no primeiro cenário podem ser executados, com a diferença de que no passo 1 uma nova versão do código-fonte da aplicação é usada para extração de *links* ICC, como ilustra a Figura 9.

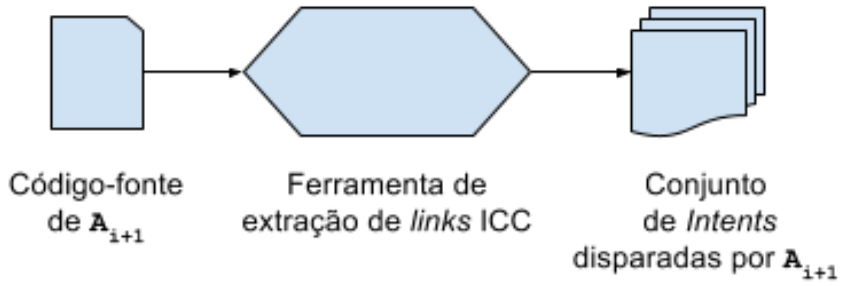


*Figura 9 - Extração de links ICC de $A_{i+1}$.*

## 3.2. PROBLEMAS IDENTIFICADOS

Dados os três cenários apresentados anteriormente dois problemas foram identificados.

### 3.2.1. Problema 1

No passo 2 do cenário 2 o *Intent Resolution* é executado com $c$ e com todos os outros itens de $C_{j+1}$. E essa não é uma abordagem interessante, pois os resultados dos testes com itens de $C_j$, onde $C_j \subset C_{j+1}$, não sofrerão alteração com a adição de $c$.

Em outras palavras, se $C_j$ é um conjunto formado por $n$ aplicativos, então $|C_j|=n$. Se de $C_j$ apenas $m$ itens tem potencial para se comunicar com $A_i$, para $m \leq n$, então $|R_k|=m$. E ao adicionarmos o aplicativo $c$ em $C_j$ teremos $c \cup C_j = C_{j+1}$ e $|C_{j+1}|=|C_j|+1$, ou $|C_{j+1}|=n+1$. Se rodarmos o *Intent Resolution* sobre $C_{j+1}$, todos os $n+1$ itens do conjunto seriam testados. Entretanto, a partir de $R_k$ é possível inferir os resultados de $n$ itens, sendo necessário apenas testar os itens recém adicionados, e neste caso, apenas $c$. Ou seja, ao invés de rodar o *Intent Resolution* $(n+1) * x$ vezes bastaria rodar $x$ vezes, sendo $x$ a quantidade de *Intents* de $A_i$. Se considerarmos que $n$=5000, por exemplo, teremos uma economia de processamento bastante considerável, já que o processamento do *Intent Resolution* aconteceria apenas $x$ vezes, e não $5001 * x$ vezes.

### 3.2.2. Problema 2

No passo 2 do cenário 3 o *Intent Resolution* opera sobre todas as *Intents* obtidas de $\mathbf{A_{i+1}}$. O problema é que nessa lista podem aparecer *Intents* que são originárias da versão anterior, $\mathbf{A_i}$, onde $\mathbf{A_i \subset A_{i+1}}$. Ou seja, os resultados para essas *Intents* antigas não mudarão, tornando desnecessários rodar novos testes com elas.

Em outras palavras, se o número de *Intents* de $\mathbf{A_i}$ é $|intents(\mathbf{A_i})|=\mathbf{x}$ e na nova versão $\mathbf{A_{i+1}}$ surgiram $\mathbf{y}$ novas *Intents*, logo $|intents(\mathbf{A_{i+1}})|=\mathbf{x}+\mathbf{y}$, para $\mathbf{A_i \subset A_{i+1}}$. Ao rodarmos o *Intent Resolution* com $\mathbf{A_{i+1}}$ e $\mathbf{C}$ o processamento acontece $(\mathbf{x}+\mathbf{y})$ $\mathbf{*\ n}$ vezes, onde $\mathbf{n}$ é quantidade de aplicativos em $\mathbf{C}$. Entretanto, se fosse possível reconhecer em $\mathbf{A_{i+1}}$ as $\mathbf{x}$ *Intents* e seus respectivos resultados remanescentes de $\mathbf{A_i}$, poderíamos economizar processamento do *Intent Resolution* sendo necessário rodá-lo apenas para as $\mathbf{y}$ novas *Intents* de $\mathbf{A_{i+1}}$. Se considerarmos que $\mathbf{x}$=1000 e $\mathbf{y}$=100, por exemplo, teremos uma economia de processamento considerável, operando apenas $\mathbf{10^2\ *\ n}$ vezes, ao invés de $\mathbf{10^5\ *\ n}$ vezes.

Uma outra situação que podemos considerar é $\mathbf{y}<0$, ou seja, na atualização de $\mathbf{A_i}$ para $\mathbf{A_{i+1}}$, ao invés de adicionadas, $\mathbf{y}$ *Intents* foram removidas. A diferença para a situação anterior é que não seria necessário processar o *Intent Resolution*, ou seja, economia de 100% em comparação à situação original do problema 2.

## 3.3. PROPOSTA

A ferramenta IccTA, já citada anteriormente, é descrita como dotada da capacidade de resolver o problema 1, isto é, ela é capaz de operar incrementalmente sobre os aplicativos [LI, 2014]. No entanto não foram encontradas evidências referentes a essa capacidade.

Então dada a carência de ferramentas capazes de resolver os dois problemas levantados anteriormente, este trabalho se propõe a desenvolver uma solução capaz de resolvê-los. Em outras palavras - dado um aplicativo Android $\mathbf{A}$ e um conjunto conhecido de aplicativos Android $\mathbf{C}$ - seus dois principais objetivos são:

I. Operar de forma incremental quando novos aplicativos forem adicionados em $\mathbf{C}$.

II. Operar de forma incremental dada uma nova versão de **A**.

O que este trabalho considera como incremental, no caso, é a ideia de combinar os resultados individuais já existentes de cada aplicativo sem a necessidade de realizar nova análise com todos quando se deseja testar uma nova aplicação adicionada ao conjunto **C**, ou quando o aplicativo **A** for atualizado.

### 3.3.1. Incremental Resolution

O *Incremental Resolution* é um algoritmo desenvolvido neste trabalho capaz de realizar múltiplas operações de *Intent Resolution* entre um dado aplicativo **A** e uma lista de outros aplicativos **C**, combinando resultados de análises antigas, novos aplicativos adicionados a **C** e o histórico de versões de **A**, evitando, sempre que possível, a reexecução de análises cujos resultados já são conhecidos.

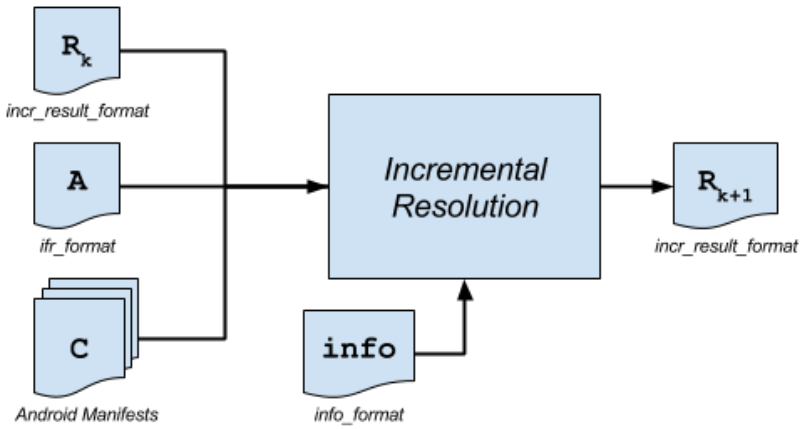


*Figura 10 - Esquema de entradas e saída do Incremental Resolution.*

Na Figura 10 é possível ver o esquema de entradas e saída do *Incremental Resolution* onde **A** representa o aplicativo que dispara as *Intents* a serem testadas, **C** representa a lista de aplicativos a serem testados com as *Intents* de **A**, **info** contém informações sobre os itens de **C**, $R_k$ corresponde ao resultado da última análise realizada e $R_{k+1}$ é o resultado da análise atual. Cada componente de entrada e saída possui formato definido, de maneira a facilitar o trabalho do algoritmo.

A entrada **A** tem o formato *ifr_format (IntentForResolution Format)* e é gerada a partir dos resultados de ferramentas como Epicc e IC3. Estas ferramentas tentam

aproximar as informações de cada *link* identificado, isso porque em alguns casos pode existir mais de um valor para o mesmo *link* ICC. Quando esses resultados são convertidos para o *ifr_format* cada combinação possível em um *link* **múltiplo** (com múltiplos valores) dá origem a um novo *link* **atômico**. O *ifr_format* é composto por uma lista de *links* **atômicos**, ou *IntentForResolution*. Cada *IntentForResolution* possui os atributos *hash, parentId, methodType, componentName, action, data, mimetype e categories,* como ilustra a Figura 11*:*

- O *hash* é gerado usando a função *hash* MD5 [Wikipédia, 2016b] a partir da concatenação dos demais atributos e é usado para identificar unicamente cada *IntentForResolution*.
- O *parentId* é uma referência ao *link* **múltiplo** original.
- O *methodType* apresenta o nome do *ICC Method* - métodos que disparam uma ICC [Li, 2015], por exemplo *"startActivity"* - associado a esta *IntentForResolution*.
- O *componentName* é o nome do componente alvo, definido quando a *IntentForResolution* é explícita.
- O *action* contém a ação da *IntentForResolution*.
- O *data* contém a *URI* especificada na *IntentForResolution*.
- O *mimetype* contém o *MIME* definido na *IntentForResolution*.
- O *categories* contém a lista de *categories* definidos na *IntentForResolution*.

```
[
  {
    "hash": "B3110A4DDDF27338E55665DD6C89077B",
    "parentId": "adblockplus_1",
    "methodType": "startActivity",
    "componentName": "",
    "action": "android.intent.action.VIEW",
    "data": "https://adblockplus.org/en/android-config#proxy",
    "mimetype": "",
    "categories": [
      "android.intent.category.DEFAULT"
    ]
  },
  {
    "hash": "98C7BFA69BCD7B7B3F0046733B840518",
    "parentId": "adblockplus_2",
    "methodType": "startActivity",
    "componentName": "",
    "action": "android.intent.action.VIEW",
    "data": "https://adblockplus.org/en/android-config",
    "mimetype": "",
    "categories": [
      "android.intent.category.DEFAULT"
    ]
  }
]
```


*Figura 11 - Exemplo de entrada* **A** *no formato ifr_format.*

A entrada $\mathbf{R_k}$ e a saída $\mathbf{R_{k+1}}$ estão no formato *incr_result_format (IncrementalResolution Result Format)*. Esse formato possui quatro valores: *app_hash, manifest_packages, intent_hashes* e *intent_results,* como ilustra a Figura 12:

- O *app_hash* é gerado usando o método de codificação Base64 (Wikipédia, 2016c) sobre a lista das *IntentForResolution (ifr_format)* de **A** usadas nas respectivas análises, na anterior para $\mathbf{R_k}$ e na atual para $\mathbf{R_{k+1}}$.
- O *manifest_packages* apresenta a lista de aplicativos de **C**, contendo duas informações, o *package*, que é único para cada aplicativo, e o *version*, um inteiro que informa a última versão daquele aplicativo com a qual **A** foi testado.
- O *intent_hashes* contém a lista do atributo *hash* das mesmas *IntentForResolution* usadas para gerar o *app_hash.*

- E por fim, o atributo *intent_result* apresenta a lista dos resultados do *Intent Resolution* das *IntentForResolution* de **A** para cada aplicativo de **C**. Para cada *IntentForResolution* de **A** o *intent_results* apresenta o respectivo *hash* (também encontrado no *intent_hashes*), e o atributo *manifests* que apresenta a mesma lista que o *manifest_packages,* com adição do atributo *matches.* Este último, contém a lista de nomes dos componentes (*component_name*) do aplicativo e o resultado (*value*) do *Intent* Resolution executado sobre o componente e a respectiva *IntentForResolution*.

O *info_format* corresponde ao formato da entrada **info**. O formato contém uma lista de informações acerca dos itens de **C**: o *name* e o *version*. O *name* é um identificador único de cada item. O *version* é um inteiro que corresponde ao valor da última versão do respectivo aplicativo em **C**. A Figura 13 mostra um trecho de exemplo de entrada **info** no formato *info_format*.


```
{
  "app_hash": "WyNoYXNoOiAyNTg5OEREMDJBQkI4OUNDQUQxMzVDMzg2REEwMEY4MwojcGFyZW5",
  "manifest_packages": [
    {
      "package": "com.github.pockethub",
      "version": 1
    }
  ],
  "intent_hashes": [
    "25898DD02ABB89CCAD135C386DA00F83",
    "25F13CCC4A5F6CF731EC746A06C7E801"
  ],
  "intent_results": [
    {
      "hash": "25898DD02ABB89CCAD135C386DA00F83",
      "manifests": [
        {
          "package": "com.github.pockethub",
          "version": 1,
          "matches": [
            {
              "component_name": "com.github.pockethub.ui.MainActivity",
              "value": false
            },
            {
              "component_name": "com.github.pockethub.ui.gist.CreateGistActivity",
              "value": true
            }
          ]
        }
      ]
```


*Figura 12 - Trecho de entrada $R_k$ no formato incr_result_format.*

```
{
  "packages": [
    {
      "name": "google_android",
      "version": 2
    },
    {
      "name": "com.fsck.k9",
      "version": 2
    },
    {
      "name": "com.github.pockethub",
      "version": 2
    }
  ]
}
```


*Figura 13 - Exemplo da entrada `info` no formato info_format.*

Os itens da entrada **C** são simplesmente cópias do *AndroidManifest.xml* de cada aplicativo contido em **C**.

A Figura 14 apresenta um pseudocódigo que descreve como o *Incremental Resolution* opera:

```
carregue A, C, R e info.
se não(estaVazio(R)) então
  se temAtualização(A, R) então
    para-cada i intentsRemovidas(A, R)
      R.remove(i)
    para-cada i intentsAdicionadas(A, R)
      para-cada c ← antigos(C, R)
        /*antigos = não foi atualizado nem adicionado agora*/
        para-cada comp ← componentes(c)
          r = IntentResolution(i, comp)
          R.adicionar(r)
  fim-se
  se temNovos(C, R) então
    para-cada i ← intents(A)
      para-cada c ← novos(C, R)
        para-cada comp ← componentes(c)
          r = IntentResolution(i, comp)
          R.adiciona(r)
  fim-se
  se temAtualizados(C, R, info) então
    para-cada i ← intents(A)
      para-cada c ← atualizados(C, R, info)
        para-cada comp ← componentes(c)
          r = IntentResolution(i, comp)
          R.adiciona(r)
  fim-se
  atualiza(R)
senão
  instancie R
  para-cada i ← intents(A)
    para-cada c ← todos(C)
      para-cada comp ← componentes(c)
        r = IntentResolution(i, c)
        R.adiciona(r)
  salva(R)
fim-se
```

*Figura 14 - Pseudocódigo do Incremental Resolution.*

Basicamente, ele carrega suas entradas e verifica se existe algum resultado **R** anterior. Não existindo um **R** anterior, um novo **R** é criado e só então é executado o *Intent Resolution* com as *Intents* de **A** e com as aplicações de **C**, então os resultados são adicionados em **R**.

Entretanto, se já existir um resultado **R** anterior, verifica-se se **A** foi atualizada e em caso positivo, os resultados das *Intents* removidas de **A** são removidos de **R**, e se existirem novas *Intents* em **A** executa-se o *Intent Resolution* com elas e com os itens antigos de **C**, e então os resultados são adicionados em **R**. Ainda no contexto de existir um **R** anterior, se existirem novos itens em **C** executa-se o *Intent Resolution* com eles e com todas as *Intents* de **A**, e então os resultados são adicionados em **R**. Por fim, se existirem itens em **C** que foram atualizados executa-se o *Intent Resolution* com eles e com todas as *Intents* de **A**, e então os resultados são adicionados em **R**.

### 3.3.2. IncR

IncR é uma ferramenta desenvolvida neste trabalho para implementar o *Incremental Resolution*. Como pode ser visto na Figura 3.9, ela tem 4 operações básicas: *setup, parseNewManifests, resolveApp* e *resolveAllApps*.

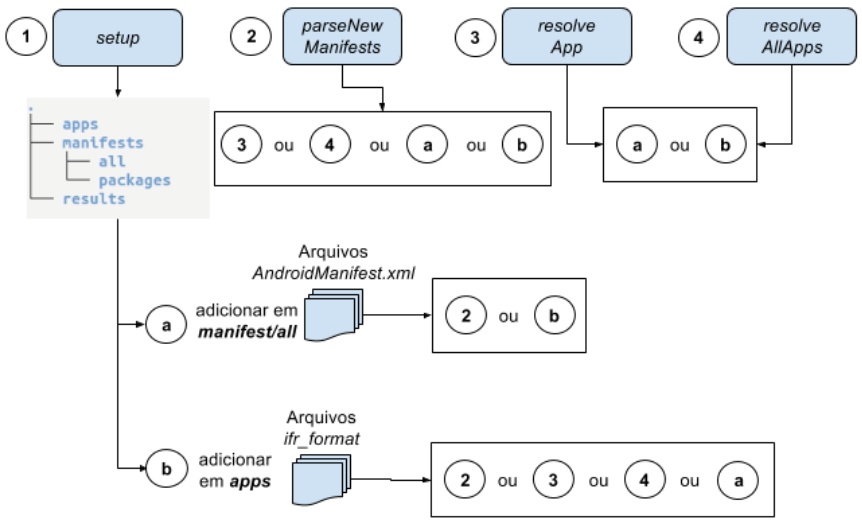


*Figura 15 - Operações do IncR, onde **a** e **b** são operações executadas pelo usuário.*

O *setup* é responsável por criar os diretórios e subdiretórios necessários ao seu funcionamento e só é preciso ser executado uma única vez. São criados 3

diretórios na raiz: *apps*, *results* e *manifests*. Como subdiretórios de *manifests* são criados os diretórios *all* e *packages.* Ainda em *manifests* é criado o arquivo *INFO.json* que corresponde a entrada **`info`** do *Incremental Resolution*. Em *apps* o usuário coloca os arquivos no formato *ifr_format* das aplicações que se deseja operar como entrada **`A`** do *Incremental Resolution*. Em *manifests/all* o usuário coloca os arquivos *AndroidManifest.xml* que se deseja incluir como item da entrada **`C`** do *Intent Resolution*.

O *parseNewManifests* é responsável por mover os novos arquivos adicionados em *manifests/all* para *manifests/packages/<package_name>,* onde *<package_name>* é o nome do pacote definido no próprio *AndroidManifest.xml.* Além disso, ele renomeia o arquivo para $v_i$*.xml,* onde **`i`** corresponde a versão daquele arquivo. Por exemplo, na primeira vez que o *AndroidManifest.xml* de um aplicativo for adicionado em *manifests/all*, ele será movido para *manifests/packages/<package_name>* e renomeado para *v1.xml*. Quando uma nova versão do *AndroidManifest.xml* do mesmo aplicativo for adicionada, ela também será movida para o mesmo diretório, sendo que agora ela será renomeada para *v2.xml*. Toda vez que um manifesto é adicionado ou atualizado, também adiciona, ou atualiza, a versão dele no arquivo *INFO.json.*

O *resolveApp* e o *resolveAllApps* executam o *Incremental Resolution*. No primeiro, o usuário escolhe um dos aplicativos em *apps,* e no segundo, todos são escolhidos, e operados um por vez. Para cada arquivo no formato *ifr_format* em *apps* que foi executado o *Incremental Resolution* um novo diretório com seu nome é criado em *results* e para cada execução do *Incremental Resolution*, se houver um novo resultado, um novo arquivo no formato *incr_result_format* é criado nesse diretório com nome $v_i$*.json*, onde **`i`** corresponde a versão do resultado. Por exemplo, na primeira vez um arquivo de resultado é criado com o nome *v1.json,* na segunda, *v2.json*, e assim por diante.

A ferramenta IncR foi implementada em Java e suas classes foram projetadas para reproduzir tanto o comportamento das quatro operações descritas anteriormente, como também todas operações associadas, como o *Incremental Resolution* e o *Intent Resolution*. A relação entre as principais classes da ferramenta pode ser vista na Figura 16, a seguir:

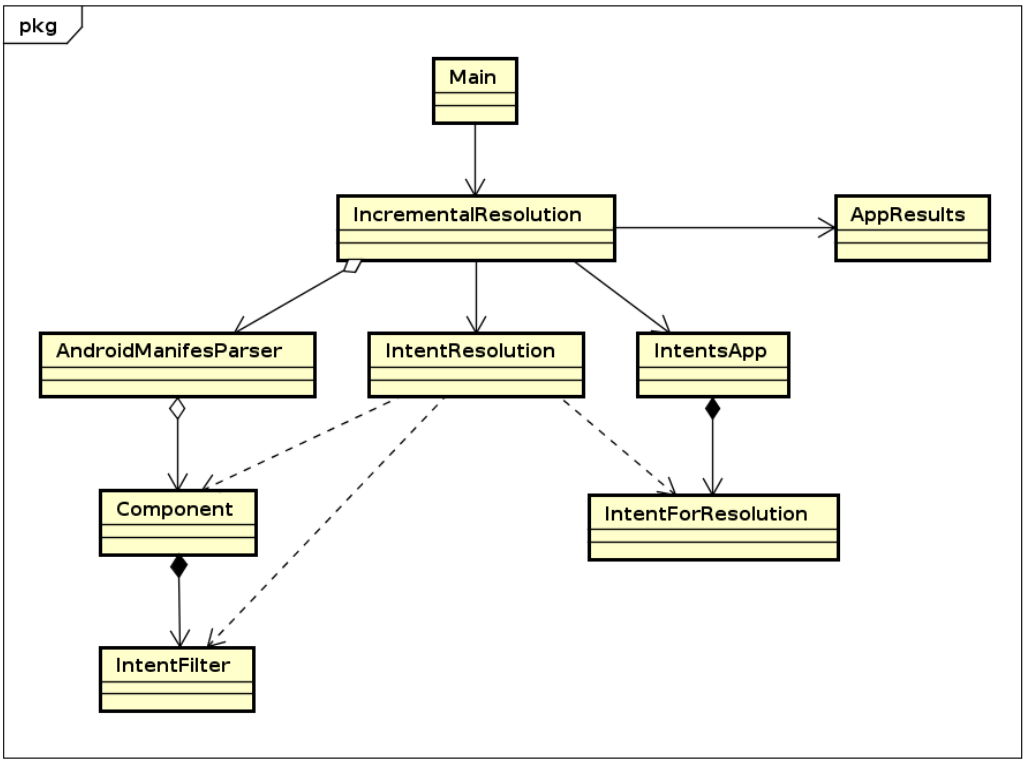


*Figura 16 - Diagrama com as principais classes do IncR.*

A classe **`Main`** implementa as operações básicas e é também responsável pelo *loop* de execução da ferramenta. As operações *resolveApp* e *resolveAllApps* utilizam a classe *IncrementalResolution* que implementa o algoritmo de mesmo nome.

A classe **`AndroidManifestParser`** corresponde a um dos itens da entrada **`C`** do *Incremental Resolution*, e é responsável por ler um arquivo *AndroidManifest.xml* e extrair seus componentes, representados pela classe **`Component`**. Esta última por sua vez é composta por uma lista de itens da classe **`IntentFilter`**.

A classe **`IntentsApp`** corresponde a entrada **`A`** do *Incremental Resolution* e é composta por uma lista de itens da classe **`IntentForResolution`***,* que por sua vez correspondem às *Intents* **atômicas,** de formato *ifr_format*, contidas em **`A`**.

A **`IncrementalResolution`** é também responsável por ler e atualizar o arquivo *INFO.json*, que corresponde a entrado **`info`** do algoritmo. Ela utiliza a classe **`IntentResolution`** para executar o *Intent Resolution* sobre os componentes de uma instância da classe **`AndroidManifestParser`** e as **`IntentForResolution`** de uma instância da classe **`IntentsApp`**.

Por fim, a classe **`AppResults`** contém os atributos descritos no formato *incr_result_format* e corresponde aos resultados gerados com a execução do IncR.

## 3.4. CONSIDERAÇÕES FINAIS

Este capítulo apresenta três cenários relacionados a identificação de comunicação entre aplicações Android. Além disso, identificamos dois problemas associados. Em contrapartida apresentamos o algoritmo *Incremental Resolution*, que se propõe a solucionar os dois problemas. E por fim, apresentamos a ferramenta IncR, que foi desenvolvida para operar o algoritmo citado.

O próximo capítulo descreve e discute os resultados dos experimentos rodados para avaliação inicial do IncR.

# 4. AVALIAÇÃO

Neste capítulo apresentamos a descrição e os resultados de dois experimentos executados com o IncR. O principal objetivo deles é avaliar o desempenho na execução do *Incremental Resolution* em comparação com a detecção não incremental de ICC, chamado nesses experimentos de *Slow Resolution*.

## 4.1. EXPERIMENTO 1

Dados uma lista de aplicações Android **A**, um conjunto de outras aplicações Android **C**, o cenário projetado para esse experimento consiste em avaliar a eficiência do *Incremental Resolution* com **sucessivas adições de novos aplicativos a C**.

Para o Experimento 1 a entrada **A** foi representada por uma lista de arquivos no formato *ifr_format* gerados a partir do código-fonte de **16** aplicativos diferentes.

Já a lista **C** de outros aplicativos era composta pelos arquivos *AndroidManifest.xml* de **1000** aplicativos diferentes.

Foi criado um *script* Linux para simular a adição de novos *AndroidManifests.xml* no diretório *manifests/all* do IncR. Primeiramente, o *script* executa a operação *setup* do IncR para criar os diretórios necessários. Em seguida ele itera sobre a lista de arquivos *AndroidManifest.xml* num determinado diretório, e a cada iteração move um deles para o diretório *manifests/all*, executa a operação *parseNewManifests* e, por fim, executa a operação *resolveAllApps*.

### 4.1.1. Resultados do Experimento 1

No Experimento 1 não foi possível obter os resultados a tempo de finalização do trabalho dos testes com mais de 500 aplicativos na abordagem *Slow*. Os resultados da abordagem *Incremental* foram obtidos normalmente. Os valores resultados estão agrupados na Tabela 1 a seguir:

*Tabela 1 - Tempo de execução do Incremental e do Slow Resolution, no Experimento 1.*

| | **10 *apps*** | **125 *apps*** | **500 *apps*** | **1000 *apps*** |
|---|---|---|---|---|
| Incremental | 13,4 seg | 4,11 min | 38,43 min | 2,11 h |
| Slow | 39,53 seg | 25,21 min | - | - |

Apesar de obtermos apenas resultados até 125 aplicativos da abordagem *Slow* é possível afirmar que o *Incremental Resolution* leva uma larga vantagem de desempenho se compararmos, por exemplo, o tempo de execução de ambas abordagens, para até 125, onde o tempo do *Slow* levou mais de quatro vezes o tempo do *Incremental*. Além do mais, o fato do *Incremental* ter terminado todos os testes, e o *Slow* não, já demonstra a sua eficiência. O gráfico da Figura 17 ilustra as curvas de crescimento de tempo de ambas as abordagens.

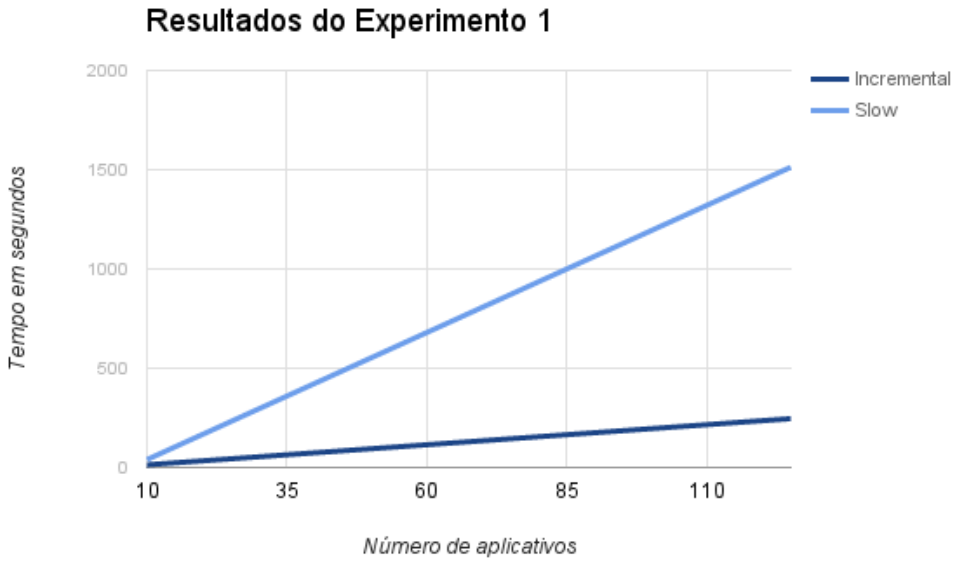


*Figura 17 - Resultados do Experimento 1*

## 4.2. EXPERIMENTO 2

Dados uma aplicação Android **A**, um conjunto de outras aplicações Android **C**, o cenário projetado para esse experimento consiste em avaliar a eficiência do *Incremental Resolution* com **sucessivas atualizações de A**.

Para esse experimento a aplicação **A** foi representada por um arquivo no formato *ifr_format* gerado a partir do código-fonte da aplicação ***VLC for Android*.** Este arquivo foi destrinchado em **37** outros arquivos do mesmo formato. Para cada *Intent* que o arquivo original continha um novo arquivo foi criado, incrementando a

quantidade de *Intents* por arquivo. Ou seja, o primeiro tinha uma *Intent*. O segundo tinha a *Intent* anterior mais uma, no total de duas *Intents*. O terceiro tinha as duas anteriores mais uma, totalizando três *Intents*. E assim por diante, de maneira que o último arquivo gerado era igual ao original. Essa estratégia foi tomada para que fossem simuladas diversas atualizações de uma mesma aplicação, onde cada arquivo gerado corresponde a uma nova atualização.

A lista $\mathbf{c}$ de outros aplicativos, composta pelos arquivos *AndroidManifest.xml*, foi distribuída em outras quatro listas de tamanhos $|\mathbf{c}_1|=10$, $|\mathbf{c}_2|=125$, $|\mathbf{c}_3|=500$ e $|\mathbf{c}_4|=1000$.

Foi criado um *script* Linux para simular a adição de novos arquivos no formato *ifr_format* de uma mesma aplicação no diretório *apps* do IncR. Primeiramente, o *script* executa a operação *setup* do IncR para criar os diretórios necessários. Em seguida ele move a lista de arquivos ($\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$ ou $\mathbf{c}_4$) *AndroidManifest.xml* de determinado diretório para *manifests/all, e então* executa a operação *parseNewManifests*. E depois ele itera sobre uma lista de arquivos no formato *ifr_format* num determinado diretório, e a cada iteração move um deles para o diretório *apps, sobrescrevendo o anterior*. Por fim, o script executa a operação *resolveAllApps*.

### 4.2.1. Resultados do Experimento 2

No Experimento 2 todos os testes executaram até o final. Os resultados estão agrupados na tabela a seguir:

*Tabela 2 - Tempo de execução do Incremental e do Slow Resolution, no Experimento 2.*

| | $\mathbf{c}_1$ | $\mathbf{c}_2$ | $\mathbf{c}_3$ | $\mathbf{c}_4$ |
|---|---|---|---|---|
| Incremental | 12,33 seg | 17,48 seg | 30,48 seg | 46,05 seg |
| Slow | 17,11 seg | 38,49 seg | 1,51 min | 2,53 min |

É possível notar a diferença de desempenho entre as duas abordagens, onde a abordagem *Incremental* se mostrou bastante eficiente quando na maioria dos resultados levou menos da metade do tempo que a abordagem *Slow*. O gráfico da Figura 18 ilustra as curvas de crescimento de tempo de ambas as abordagens.

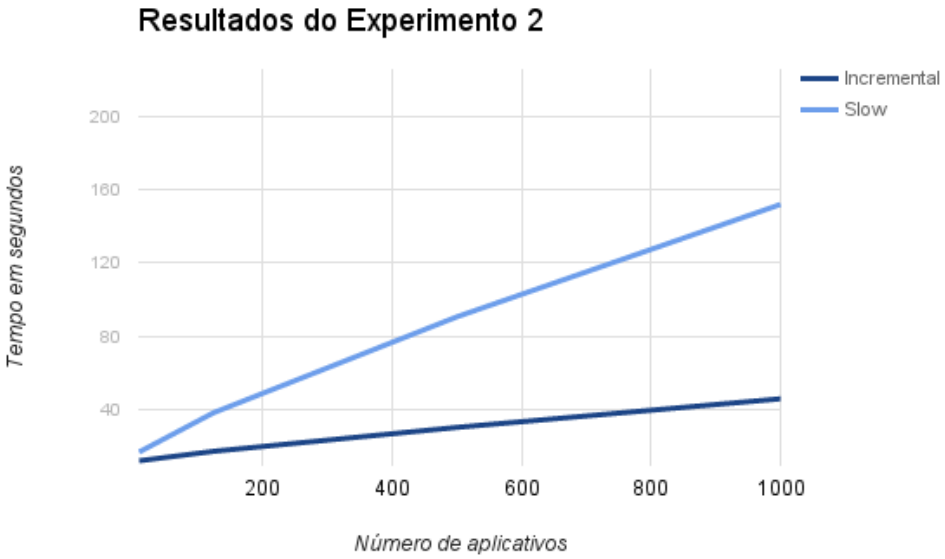


*Figura 18 - Resultados do Experimento 2*

Comparando os dois experimentos, uma possível causa para que no segundo os testes com a abordagem *Slow* finalizaram em tempo hábil e no primeiro não, é a quantidade de aplicativos usados na entrada **A**. Como no primeiro experimento usamos 16 aplicativos em **A**, a cada iteração do *Slow Resolution* todos os *AndoridManifest.xml* adicionados eram novamente testados para cada uma das *IntentsForResolution* de cada um dos 16 aplicativos de **A**. Enquanto que no segundo experimento apenas uma aplicação foi usada em **A**, mesmo quando simulamos as atualizações deste aplicativo, apenas as *IntentForResolution* dele eram testadas com todos os *AndroidManifest.xml,* ou seja, os testes com os manifestos só se repetiram com as *IntentForResolution* de uma aplicação. Entretanto, carece de uma investigação mais detalhada de outros possíveis fatores que influenciaram esse comportamento.

## 4.3 CONSIDERAÇÕES FINAIS

Este capítulo apresenta dois experimentos executados sobre o IncR com objetivo de avaliar o desempenho da sua abordagem incremental na identificação de ICC, em comparação com a abordagem padrão, chamada nos experimentos de *Slow Resolution.* No primeiro experimento não foi possível obter todos os resultados do *Slow,* pois suas execuções não terminaram em tempo hábil para finalização deste trabalho. Enquanto que para os valores obtidos o *Incremental Resolution* mostrou-se de longe mais eficiente com pelo menos um quarto do tempo do *Slow,* executados com até 125 aplicativos. Já no experimento 1, todos as execuções finalizaram em

tempo hábil e ainda assim a abordagem *Incremental* obteve resultados que demostram sua eficiência, em geral levando metade do tempo do *Slow,* nos testes a partir de 125 aplicativos.

# 5. CONCLUSÃO

Este trabalho apresentou o algoritmo *Incremental Resolution*, desenvolvido a partir da necessidade de identificar incrementalmente a comunicação entre aplicativos Android. Também apresentou a ferramenta IncR, que implementa o algoritmo citado. Inicialmente, foram introduzidos conceitos importantes para o entendimento da plataforma Android e do próprio trabalho como um todo. Tais como arquitetura, componentes, *Intent*, o Inter-component communication (ICC), entre outros. Alguns cenários envolvendo a detecção de ICC foram levantados e foi identificado que existe uma certa carência de ferramentas capazes de realizar a detecção incremental de ICC, visto que das ferramentas existentes, apenas uma diz ser capaz de fazê-lo, mas não demonstra evidências sobre tal capacidade.

Então, diante da carência de ferramenta capaz de operar incrementalmente, foram propostos: o algoritmo *Incremental Resolution* e a ferramenta que o implementa, IncR. O *Incremental Resolution*, por sua vez, possui entradas e saída com formatos específicos, também detalhados neste trabalho. Foram apresentados alguns detalhes da implementação do IncR, como sua arquitetura, e a relação das suas classes e operações com o *Incremental Resolution*.

Por fim, foram realizados dois tipos experimentos sobre o IncR com o propósito de analisar sua performance na execução do *Incremental Resolution*. Seus resultados dão evidência inicial do desempenho superior da ferramenta quando comparada com a abordagem padrão.

## 5.1. TRABALHOS RELACIONADOS

Algumas ferramentas como Epicc [Octeau et al, 2013] e IC3 [Octeau at al, 2015], assim como outras (SOUZA, 2016), são capazes de extrair *links* ICC das aplicações a partir de análises estáticas, e limitam-se a apenas isso. Entretanto, este é um primeiro passo tomado em direção à detecção de ICC. Pois, elas podem ser combinadas com outras ferramentas, como IccTA [LI, 2014]. Esta por sua vez, de posse dos *links* ICC, se propõe a encontrar a comunicação entre componentes e identificar possíveis pontos de vulnerabilidades. Inclusive IccTA, assim como o IncR,

opera de forma incremental, no entanto, não foram encontrados detalhes acerca desta capacidade.

## 5.2. TRABALHOS FUTUROS

Considerando os dois principais objetos desenvolvidos neste trabalho, o algoritmo *Incremental Resolution* e a ferramenta IncR, algumas sugestões de possíveis evoluções são:

- **Uso de Banco de Dados**: Na versão atual da ferramenta IncR todos os históricos de versões das entradas e dos resultados são gerenciados em arquivos. E esta abordagem pode caracterizar uma certa perda de desempenho em sua execução, além do possível consumo excessivo de espaço em disco. Em um trabalho futuro pode-se buscar a comparação entre o atual desempenho da ferramenta com o desempenho de implementações alternativas que usam diferentes tipos de banco de dados.
- **Avaliação do *Intent Resolution*:** Neste trabalho o mecanismo de *Intent Resolution* foi implementado como uma sub-operação do *Incremental Resolution*. No entanto, não tem por objetivo avaliar a precisão do *Intent Resolution* implementado, ou seja, restringe-se em apenas avaliar a eficiência da ferramenta IncR. Como trabalho futuro, pode-se avaliar a precisão de identificação de ICC e propor melhorias na implementação do *Intent Resolution*.
- **Gerenciamento de Componentes:** Uma das limitações do *Incremental Resolution* é que ele não é capaz de realizar gerenciamento no nível dos componentes da aplicação, portanto, ele apenas identifica que um determinado *AndroidManifest.xml* foi atualizado. Em outras palavras, quando um arquivo de manifesto é atualizado, o *Incremental Resolution* não consegue detectar quais componentes são novos, quais foram removidos e nem quais não sofreram nenhuma modificação. Como um trabalho futuro, pode-se evoluir o *Incremental Resolution* para ser capaz de gerenciar também os componentes das aplicações e avaliar o impacto no seu desempenho.
- **Mapa de fluxo de dados:** Com os resultados gerados com a execução do *Incremental Resolution*, num trabalho futuro pode-se buscar combiná-los para

construir um mapa do fluxo de um determinado dado durante a comunicação entre diversos aplicativos, e dessa forma aumentar a capacidade de mitigar possíveis vazamentos de dados por aplicativos maliciosos.

# 6. REFERÊNCIAS BIBLIOGRÁFICAS

**Android.** Disponível em <https://pt.wikipedia.org/wiki/Android>. Acesso em: 11 de Julho de 2016.

**Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015**. Disponível em: <http://www.gartner.com/newsroom/id/3215217>. Acesso em: 11 de Julho de 2016.

**Number of available applications in the Google Play Store from December 2009 to February 2016**. Disponível em: <http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>. Acesso em: 11 de Julho de 2016.

**The Android Source Code**. Disponível em: <https://source.android.com/source/index.html>. Acesso em: 11 de Julho de 2016

**ART Features.** Disponível em: <https://source.android.com/devices/tech/dalvik/index.html#features>. Acesso em: 11 de Julho de 2016

**Hardware abstract layer (HAL).** Disponível em: <https://source.android.com/devices/index.html#Hardware%20Abstraction%20Layer>. Acesso em: 11 de Julho de 2016.

BORDIN, M. V.; Introdução a Arquitetura Android**. In: VII Workshop de Tecnologias da Informação SETREM**, 7., 2012. Três de Maio. Anais… Três de Maio: Sociedade Educacional Três de Maio, 2012 p. 3.

**Application Fundamentals**. Disponível em: <https://source.android.com/source/index.html>. Acesso em: 11 de Julho de 2016.

**App Componets**. Dispoível em: <https://developer.android.com/guide/components/fundamentals.html#Components>. Acesso em: 11 de Julho de 2016.

**Intent Types**. Disponíivel em: <https://developer.android.com/guide/components/intents-filters.html#Types>. Acesso em: 11 de Julho de 2016.

OCTEAU, D., MCDANIEL, P., JHA, S. et al. Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis. **In: 22nd USENIX Security Symposium**. Washington, D.C, 2013 p. 1.

OCTEAU, D., LUCHAUP, D., DERING, M. et al. **Composite Constant Propagation: Application to Android Inter-Component Communication Analysis**. ICSE, 2015. Florence, Itália.

SOUZA, V. C. P. **Uma ferramenta leve de análise para descoberta estática de comunicações entre componentes de aplicações Android**. Trabalho de Graduação - Centro de Informática, Universidade Federal de Pernambuco, 2016. Recife, Brasil.

**Receiving an Implicit Intent.** Disponível em: <https://developer.android.com/guide/components/intents-filters.html#Receiving>. Acesso em: 11 de Julho de 2016.

**Intent Resolution.** Disponível em: <https://developer.android.com/guide/components/intents-filters.html#Resolution>. Acesso em: 11 de Julho de 2016.

LU, L., LI, Z., WU, Z., LEE, W., JIANG, G., **CHEX: statically vetting android apps for component hijacking vulnerabilities**. ACM CCS, 2012. Nova Iorque, EUA.

LI, L., BARTEL, A., KLEIN, J., TRAON, Y. L., ARZT, S. et al. **I know what leaked in your pocket: uncovering privacy leaks on Android Apps with Static Taint Analysis**. 2014.

**MD5**. Disponível em: <https://pt.wikipedia.org/wiki/MD5>. Acesso em: 11 de Julho de 2016.

LI, L., BARTEL, A., BISSYANDÉ, T. F., KLEIN, J., TRAON, Y. L., ARZT, S. et al. **IccTA: Detection Inter-Component Privacy Leaks in Android Apps**. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. 2015 p. 2.

**Base64**. Disponível em: <https://pt.wikipedia.org/wiki/Base64>. Acesso em: 11 de Julho de 2016.

**Late binding vai muito bem, obrigado.** Disponível em <http://www.devmedia.com.br/late-binding-vai-muito-bem-obrigado/4435>. Acesso em: 11 de Julho de 2016.

**App Manifest.** Disponível em: <https://developer.android.com/guide/topics/manifest/manifest-intro.html>. Acesso em: 11 de Julho de 2016.

**Google says there are now 1.4 billion active Android devices worldwide**. Disponível em: <http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide>. Acesso em: 15 de Julho de 2016.