ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

NGUYỄN TẤN CẦM

PHÁT HIỆN NGUY CƠ THẤT THOÁT THÔNG TIN TRÊN ĐIỆN THOẠI DI ĐỘNG ANDROID

Chuyên ngành: **Công nghệ thông tin**
Mã số: **62 48 02 01**

TÓM TẮT LUẬN ÁN TIẾN SĨ CÔNG NGHỆ THÔNG TIN

Thành phố Hồ Chí Minh - 2020

Công trình được hoàn thành tại:

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN -
ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

Người hướng dẫn khoa học:

TS. Nguyễn Anh Tuấn
TS. Phạm Văn Hậu

Phản biện 1: .....
Phản biện 2: .....
Phản biện 3: .....

Luận án sẽ được bảo vệ trước Hội đồng chấm luận án cấp Trường tại:
.....
.....
vào lúc … giờ … ngày … tháng … năm … .

Có thể tìm hiểu luận án tại:

- Thư viện Quốc gia Việt nam.
- Thư viện ĐHQG-HCM.
- Thư viện Trường Đại học Công nghệ Thông tin – Đại học Quốc gia Tp. Hồ Chí Minh.

MỤC LỤC

GIỚI THIỆU 2

Chương 1. TỔNG QUAN 3

1.1 Giới thiệu tóm tắt về công trình nghiên cứu 3

1.2 Mục đích, đối tượng và phạm vi nghiên cứu 3

1.3 Ý nghĩa khoa học và thực tiễn của đề tài 4

1.4 Bố cục của luận án 4

Chương 2. TỔNG QUAN TÌNH HÌNH NGHIÊN CỨU 5

2.1 Các định nghĩa được sử dụng trong luận án 5

2.2 Tổng quan về hướng nghiên cứu phát hiện thất thoát dữ liệu nhạy cảm trong các thiết bị sử dụng hệ điều hành Android 6

2.3 Các thách thức hiện tại 11

Chương 3. HƯỚNG TIẾP CẬN CỦA LUẬN ÁN 13

3.1 Lý do lựa chọn hướng tiếp cận của luận án 13

3.2 Dữ liệu thử nghiệm 14

3.2.1 Bộ dữ liệu thử nghiệm tình huống gây thất thoát thông tin nhạy cảm qua liên ứng dụng 14

3.2.2 Bộ dữ liệu thử nghiệm cho firmware Android tùy biến 15

3.3 Các đề xuất của luận án 16

3.3.1 Đề xuất phương pháp cải thiện độ chính xác trong việc phát hiện thất thoát thông tin nhạy cảm qua liên ứng dụng 16

3.3.2 Đề xuất phương pháp phát hiện thất thoát thông tin nhạy cảm trong thành phần Application Framework của hệ điều hành Android 35

Chương 4. KẾT LUẬN 39

4.1 Các kết quả đạt được 39

4.2 Các công bố trong luận án 39

4.2.1 Các công bố chính của luận án 39

4.2.2 Các công bố phụ liên quan đến luận án 40

4.2.3 Giải pháp hữu ích liên quan đến luận án 40

4.2.4 Các đề tài đã tham gia 40

4.3 Hạn chế và hướng phát triển của luận án 41

4.3.1 Hạn chế 41

4.3.2 Hướng phát triển của luận án 41

GIỚI THIỆU

Điện thoại thông minh được sử dụng phổ biến trong thời đại ngày nay. Trong đó, điện thoại sử dụng hệ điều hành Android được sử dụng nhiều hơn so với các điện thoại sử dụng các hệ điều hành khác. Nguy cơ bảo mật có thể tồn tại ở nhiều thành phần khác nhau của điện thoại sử dụng hệ điều hành Android, ví dụ như các ứng dụng cài sẵn, thành phần Application Framework, thậm chí là ở thành phần Linux Kernel. Phát hiện thất thoát thông tin nhạy cảm trong các điện thoại sử dụng hệ điều hành Android là một hướng nghiên cứu được quan tâm. Các nghiên cứu hiện tại chủ yếu tập trung phân tích nguy cơ bảo mật trong các ứng dụng cài sẵn trên điện thoại sử dụng hệ điều hành Android. Trong quá trình phân tích nguy cơ gây rò rỉ thông tin nhạy cảm trong các ứng dụng cài sẵn, các nghiên cứu hiện tại chủ yếu phân tích trên ứng dụng đơn. Trong luận án này, chúng tôi đề xuất các phương pháp cải thiện độ chính xác trong việc phát hiện thất thoát thông tin nhạy cảm qua liên ứng dụng và phương pháp phát hiện thất thoát thông tin nhạy cảm trong thành phần Application Framework của hệ điều hành Android. Kết quả thực hiện của luận án cho thấy các đề xuất trong luận án có thể được sử dụng để rút trích các thành phần khác nhau trong hệ điều hành Android của nhiều hãng điện thoại khác nhau và có thể phân tích nguy cơ gây thất thoát thông tin nhạy cảm trong hai thành phần ở mức cao nhất của hệ điều hành Android là ứng dụng và Application Framework.

Trong quá trình thực hiện luận án này, nghiên cứu sinh công bố 09 bài báo, trong đó 02 bài tạp chí thuộc danh mục ISI (với IF=2.04), 02 bài tạp chí trong nước thuộc danh mục tính đến 01 điểm của Hội đồng chức danh giáo sư nhà nước, và 05 bài báo đăng trên kỷ yếu hội nghị khoa học quốc tế.

Bên cạnh đó, trong quá trình thực hiện luận án, nghiên cứu sinh cùng tập thể cán bộ hướng dẫn tham gia thực hiện các đề tài nghiên cứu khoa học và đăng ký các giải pháp hữu ích liên quan trực tiếp đến các công việc chính của luận án. Cụ thể nghiên cứu sinh và cán bộ hướng dẫn tham gia hai đề tài nghiên cứu khoa học. Trong đó, một đề tài nghiên cứu khoa học cấp Sở khoa học và công nghệ TP.HCM, một đề tài nghiên cứu khoa học thuộc loại đề tài B, Đại học Quốc gia TP.HCM.

Ngoài ra, nghiên cứu sinh và cán bộ hướng dẫn đăng ký hai giải pháp hữu ích theo hai quyết định của Cục Sở hữu trí tuệ, Bộ Khoa học và Công nghệ.

Chương 1. TỔNG QUAN

1.1 Giới thiệu tóm tắt về công trình nghiên cứu

Trong luận án này, nghiên cứu sinh đề xuất một số phương pháp cải thiện độ chính xác của việc phát hiện thất thoát thông tin nhạy cảm trong thiết bị sử dụng hệ điều hành Android. Cụ thể như sau:

Thứ nhất, luận án đề xuất phương pháp cải thiện độ chính xác của việc phát hiện thất thoát thông tin nhạy cảm trong các ứng dụng Android. Cụ thể phương pháp đề xuất cho phép mở rộng các loại giao tiếp liên ứng dụng, kiểm tra khả năng tồn tại của các giao tiếp liên ứng dụng, tiết kiệm thời gian trong quá trình phân tích động. Một phần nội dung của đóng góp này được đăng trong tạp chí Cluster Computing (Springer, ISI, IF: 2.04) [CB1, CB2], kỷ yếu hội nghị ICISA2016 (Springer) [CB3] và ICCSN2017 (IEEE) [CB5].

Thứ hai, luận án đề xuất phương pháp áp dụng kỹ thuật phân tích luồng dữ liệu trong ứng dụng Android để phát hiện thất thoát thông tin nhạy cảm trong thành phần Application Framework trong firmware Android. Một phần nội dung của phương pháp này được trình bày tại IEEE MDM2017 [CB4], FAIR2017 [CB6]

Thứ ba, đóng góp về mặc kỹ thuật trong việc phát triển các công cụ phục vụ cho quá trình thực hiện các đề xuất của luận án. Mặc dù, đóng góp thứ ba mang tính kỹ thuật là chủ yếu, tuy nhiên công sức mà chúng tôi bỏ ra là đáng ghi nhận trong quá trình thực hiện luận án.

Bên cạnh đó, một đóng góp phụ khác của luận án là xây dựng bộ dữ liệu thử nghiệm có số kịch bản chứa đầy đủ trường hợp gây thất thoát thông tin nhạy cảm hơn. Đây được xem là đóng góp giá trị cho cộng đồng nghiên cứu trong lĩnh vực này. Bộ dữ liệu thử nghiệm này nhận được ý kiến phản hồi tích cực từ các nhóm nghiên cứu liên quan. Một phần nội dung của đóng góp này được trình bày tại hội nghị ICCAI2019 [CB8] (kỷ yếu xuất bản bởi ACM).

1.2 Mục đích, đối tượng và phạm vi nghiên cứu

Mục đích nghiên cứu: Đề xuất các phương pháp cải thiện độ chính xác trong việc phát hiện thất thoát thông tin nhạy cảm trong các thiết bị sử dụng hệ điều hành Android. Cụ thể luận án phân tích nhiều thành phần khác nhau của hệ điều hành Android bằng cách tập trung phân tích luồng thông tin nhạy cảm trong các ứng dụng cài sẵn và thành phần Application Framework của hệ điều hành Android. Kết quả nghiên cứu có thể được sử dụng cho việc gom nhóm các ứng dụng có khả năng kết hợp gây thất thoát thông tin nhạy cảm để từ đó người dùng biết được có nên cài một ứng dụng nào đó khi kết hợp nó với các ứng dụng khác hay không. Kết quả nghiên cứu cũng có thể được sử dụng để đánh giá khả năng gây thất thoát thông tin nhạy cảm trong các thiết bị sử dụng hệ điều hành Android.

Đối tượng nghiên cứu: Luận án tập trung vào việc phân tích hai thành phần chính của hệ điều hành Android là các ứng dụng và thành phần Application Framework.

Phạm vi nghiên cứu:

Luận án tập trung vào việc phát hiện thất thoát thông tin nhạy cảm thông qua việc xác định sự tồn tại của luồng dữ liệu gây thất thoát thông tin nhạy cảm trong hai thành phần chính của hệ điều hành Android là các ứng dụng và Application Framework. Trong phạm vi của luận án này chúng tôi dùng FlowDroid [1], IC3 [2] và danh sách hàm Source-Sink cập nhật từ SuSi [3] để phát hiện luồng dữ liệu gây thất thoát thông tin nhạy cảm. Về mặt kỹ thuật, danh sách các hàm Source-Sink này có thể bổ sung trong tương lai mà không cần thiết phải thay đổi phương pháp đề xuất. Luồng dữ liệu gây thất thoát thông tin nhạy cảm có thể trong một ứng dụng hoặc có thể đi qua nhiều ứng dụng khác nhau. Trong trường hợp, luồng dữ liệu qua nhiều ứng dụng, chúng tôi phân tích các hàm thực hiện chức năng giao tiếp liên ứng dụng để xác định chuỗi các ứng dụng mà luồng thông tin đi qua. Luận án chỉ xét các giao tiếp liên ứng dụng sau:

- Giao tiếp thông qua Intent (bao gồm các thành phần Activity, Service, Broadcast receiver).
- Giao tiếp thông qua Content Provider.
- Giao tiếp thông qua việc đọc và ghi trên cùng một tập tin chia sẻ.

Luận án không xét các kênh giao tiếp khác như dùng chung bộ nhớ RAM, dùng chung quy tắc định nghĩa trước – covert chanel (Ví dụ: ứng dụng thứ nhất chỉnh độ sáng màn hình, ứng dụng thứ hai đọc giá trị độ sáng màn hình,…),…

1.3 Ý nghĩa khoa học và thực tiễn của đề tài

Các nghiên cứu liên quan hiện tại chủ yếu tập trung phân tích khả năng gây rò rỉ thông tin nhạy cảm trong từng ứng dụng đơn. Cần có nhiều nghiên cứu hơn trong hướng nghiên cứu liên quan đến kỹ thuật phân tích kết hợp giữa phân tích tĩnh và phân tích động trên nhóm ứng dụng để đánh giá khả năng gây thất thoát thông tin nhạy cảm. Luận án đề xuất các kỹ thuật phân tích tĩnh liên ứng dụng, động liên ứng dụng và tĩnh động kết hợp liên ứng dụng nhằm giải quyết các hạn chế của các nghiên cứu liên quan hiện tại.

Việc đánh giá nguy cơ gây thất thoát thông tin nhạy cảm trên các thiết bị Android không những tiến hành phân tích luồng thông tin nhạy cảm liên ứng dụng trong các ứng dụng cài sẵn mà còn trong các thành phần khác như Application Framework trong các hệ điều hành Android. Luận án đề xuất phương pháp cải thiện độ chính xác của việc phát hiện thất thoát thông tin nhạy cảm trong các thiết bị Android bằng cách phân tích luồng dữ liệu nhạy cảm trong hai thành phần chính của hệ điều hành Android là các ứng dụng và Application Framework.

Kết quả nghiên cứu của luận án có thể được áp dụng trong việc gom nhóm các ứng dụng có nguy cơ kết hợp để thực hiện việc gây thất thoát thông tin. Việc gom nhóm này cho phép người dùng biết được có nên cài một ứng dụng nào đó chung với các ứng dụng khác hay không. Bên cạnh đó, kết quả của luận án có thể được sử dụng để đánh giá các firmware Android tùy biến trước khi nạp chúng vào điện thoại.

1.4 Bố cục của luận án

Nội dung của luận án được bố cục gồm 5 chương, tài liệu tham khảo và phụ lục.

Chương 1: Mở đầu.

Chương 2: Tổng quan tình hình nghiên cứu.

Chương 3: Cơ sở lý luận và giả thuyết khoa học.

Chương 4: Hướng tiếp cận của luận án.

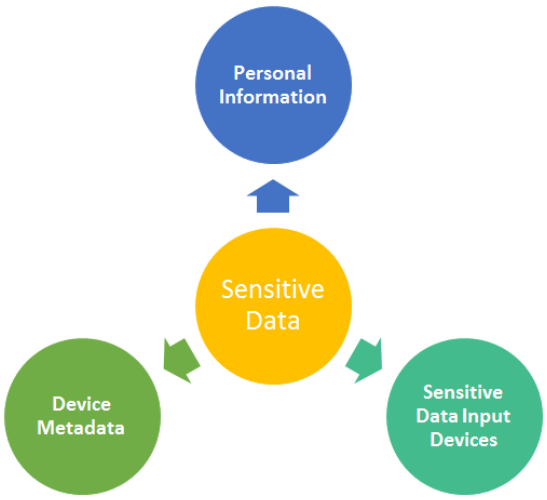
Chương 5: Kết luận

Chương 2. TỔNG QUAN TÌNH HÌNH NGHIÊN CỨU

2.1 Các định nghĩa được sử dụng trong luận án

Định nghĩa 1: Dữ liệu nhạy cảm là các loại dữ liệu cần được bảo vệ, và cần phải được sự giám sát của người dùng trước khi gửi ra khỏi thiết bị.

Theo [4], dữ liệu nhạy cảm trong các điện thoại sử dụng hệ điều hành Android được gom thành ba nhóm chính: Personal Information, Sensitive Data Input Devices, Device Metadata. Trong đó, Personal Information là các dữ liệu cá nhân, ví dụ như: danh bạ, lịch, …; Sensitive Data Input Devices là các dữ liệu từ các cảm biến (sensor) của thiết bị như: GPS, Microphone, Camera,...; Device Metadata là các thông tin của thiết bị như số IMEI, thông tin mạng, số điện thoại,...



Hình 1. Ba nhóm dữ liệu nhạy cảm trong điện thoại sử dụng hệ điều hành Android

Định nghĩa 2: Nguồn nhạy cảm (*Sensitive Source*) là các lời gọi hàm dùng để đọc các dữ liệu nhạy cảm.

Ví dụ hàm *getDeviceId* (Đoạn mã 1) dùng để đọc số IMEI của điện thoại là một trong những dữ liệu nhạy cảm thuộc nhóm Device Metadata.

Đoạn mã 1. Hàm đọc số IMEI của điện thoại

```
<com.android.internal.telephony.gsm.GSMPhone: java.lang.String
getDeviceId()> (UNIQUE_IDENTIFIER)
```

Định nghĩa 3: Đích nguy hiểm (*Critical Sink*) là các lời gọi hàm dùng để gửi các dữ liệu ra khỏi thiết bị.

Ví dụ hàm *sendTextMessage* (Đoạn mã 2) dùng để gửi thông tin ra khỏi thiết bị thông qua tin nhắn SMS.

Đoạn mã 2. Hàm gửi tin nhắn

```
<android.telephony.gsm.SmsManager: void
sendTextMessage(java.lang.String,java.lang.String,java.lang.String,a
ndroid.app.PendingIntent,android.app.PendingIntent)> (SMS_MMS)
```

Định nghĩa 4: *Exit Point* là các lời gọi hàm dùng để gửi dữ liệu ra khỏi ứng dụng.

Ví dụ hàm *strartActivity* là hàm dùng để gửi dữ liệu từ ứng dụng này sang ứng dụng khác thông qua Intent.

Đoạn mã 3. Hàm startActivity

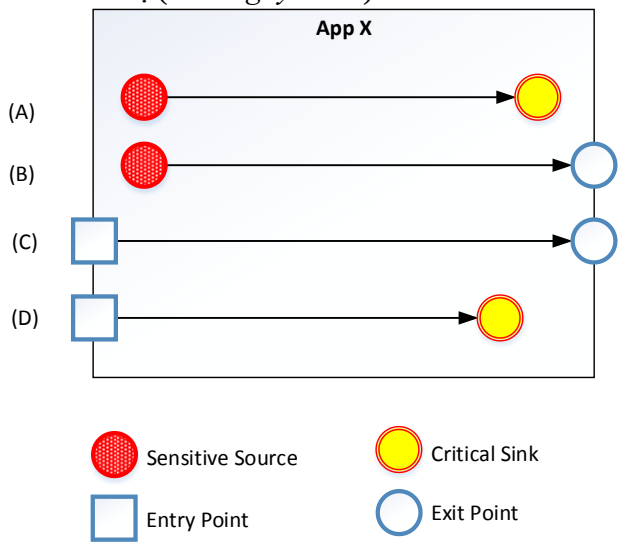
```
 <android.support.v7.app.AppCompatActivity: void
startActivity(android.content.Intent)>
```

Định nghĩa 5: *Entry Point* là các lời gọi hàm được các ứng dụng sử dụng để đọc dữ liệu từ bên ngoài của ứng dụng.

Ví dụ hàm `<android.support.v7.app.AppCompatActivity: android.content.Intent getIntent()>` dùng để đọc thông tin trong Intent.

Định nghĩa 6: Luồng dữ liệu nhạy cảm (*Sensitive Data Flow*) là luồng thông tin đi từ *nguồn nhạy cảm* đến *đích nguy hiểm*.

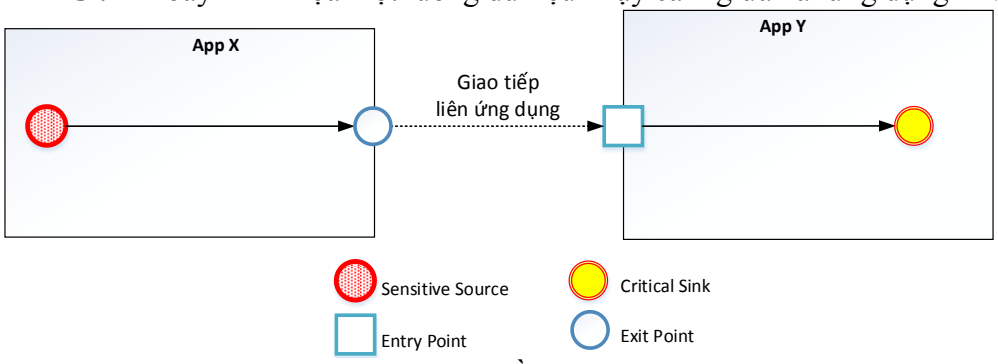
Hình 2 trình bày các loại luồng dữ liệu trong một ứng dụng Android. Trong đó, luồng dữ liệu (A) là luồng dữ liệu nhạy cảm. Luồng dữ liệu (B) đọc dữ liệu nhạy cảm (*nguồn nhạy cảm)* sau đó gửi đến các hàm cho phép gửi dữ liệu ra khỏi ứng dụng (*Exit Point*). Luồng dữ liệu (C) nhận dữ liệu từ các ứng dụng khác bằng các hàm *Entry Point* và gửi dữ liệu đến các hàm *Exit Point*. Trong khi đó, luồng (D) cho phép nhận dữ liệu từ các hàm *Entry Point* và gửi đến các hàm gây rò rỉ dữ liệu ra khỏi thiết bị (*đích nguy hiểm*).



Hình 2. Các loại luồng dữ liệu trong một ứng dụng

Định nghĩa 7: Luồng dữ liệu nhạy cảm liên ứng dụng (*Inter-Application Sensitive Data Flow*) là luồng dữ liệu nhạy cảm có nguồn bắt đầu ở một ứng dụng và đích kết thúc ở một ứng dụng khác.

Hình 3 trình bày minh họa một luồng dữ liệu nhạy cảm giữa hai ứng dụng X và Y.



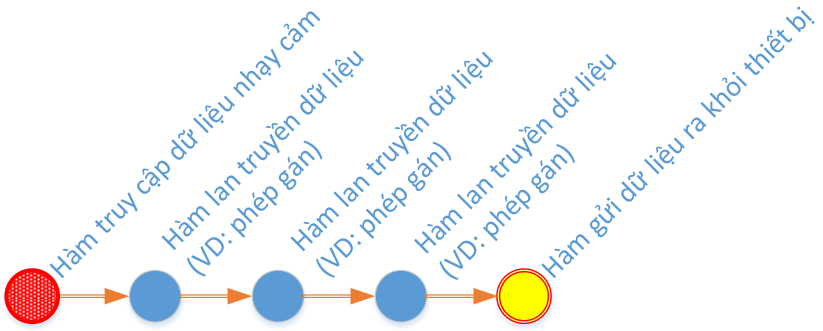
Hình 3. Hình minh họa luồng dữ liệu nhạy cảm liên ứng dụng

2.2 Tổng quan về hướng nghiên cứu phát hiện thất thoát dữ liệu nhạy cảm trong các thiết bị sử dụng hệ điều hành Android

Theo thống kê của Gartner [5] thiết bị di động luôn chiếm thị phần cao. Theo một nghiên cứu khác từ IDC [6], Android là hệ điều hành điện thoại di động phổ biến nhất so với các hệ điều hành khác. Theo thống kê của Symantec [7], có đến 94% các họ mã độc mới trên điện thoại di động nhắm đến hệ điều hành Android. Các thống kê này là động lực cho việc triển khai các nghiên cứu liên quan đến việc phân tích bảo mật trên các thiết bị sử dụng hệ điều hành

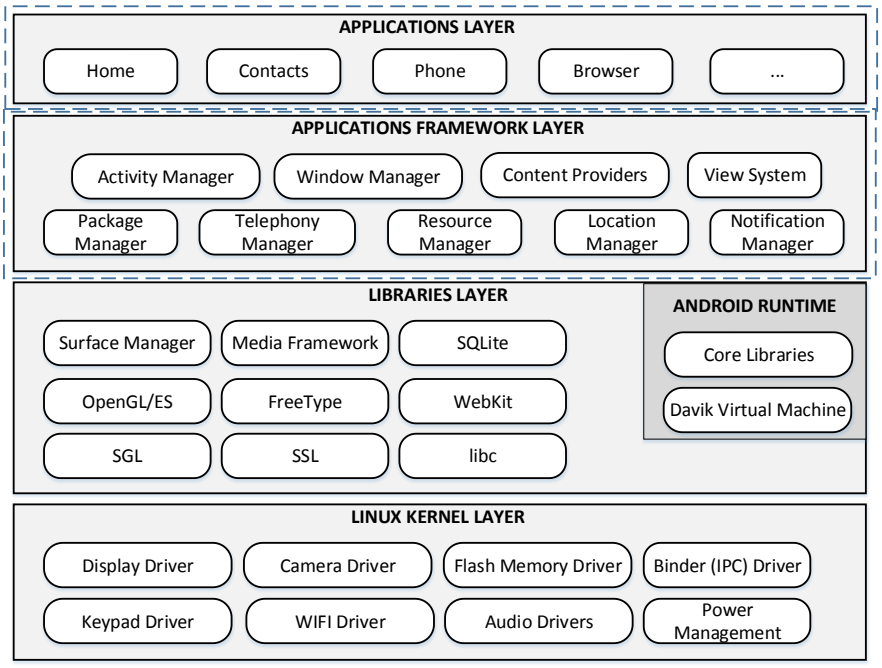
Android. Cũng theo thống kê của Symantec [7], trong các nguy cơ bảo mật trên hệ điều hành Android, đánh cắp dữ liệu nhạy cảm chiếm tỷ lệ cao nhất với 36%. Qua các thống kê này, chúng ta thấy rằng việc phát hiện thất thoát dữ liệu nhạy cảm trên các điện thoại Android là cần thiết.

Thất thoát thông tin nhạy cảm là hành động làm rò rỉ dữ liệu nhạy cảm ra khỏi thiết bị. Tức là tồn tại các luồng dữ liệu nhạy cảm theo *Định nghĩa 6* hoặc *Định nghĩa 7*. Hình 4 mô tả trường hợp dữ liệu nhạy cảm bị thất thoát ra khỏi thiết bị.



Hình 4. Minh họa luồng dữ liệu nhạy cảm

Hệ điều hành Android có nhiều thành phần khác nhau (Hình 5), như: Applications, Application Framework, Libraries, Linux Kernel,… Về mặt lý thuyết, nguy cơ gây thất thoát thông tin nhạy cảm có thể tồn tại ở bất kỳ thành phần nào của hệ điều hành Android. Như vậy, việc đánh giá nguy cơ gây thất thoát thông tin nhạy cảm trong hệ điều hành Android có thể được thực hiện bằng cách phân tích các thành phần này. Tuy nhiên, trong thực tế, theo danh sách 47 nghiên cứu liên quan chính mà nghiên cứu sinh tham khảo thì có đến hơn 85% các nghiên cứu tập trung vào việc phân tích các ứng dụng Android, tức là đối tượng nghiên cứu của các nghiên cứu này là thất thoát dữ liệu nhạy cảm trong thành phần Applications. Trong khi đó, chỉ có chưa đến 15% các nghiên cứu liên quan chọn đối tượng nghiên cứu là Android firmware.

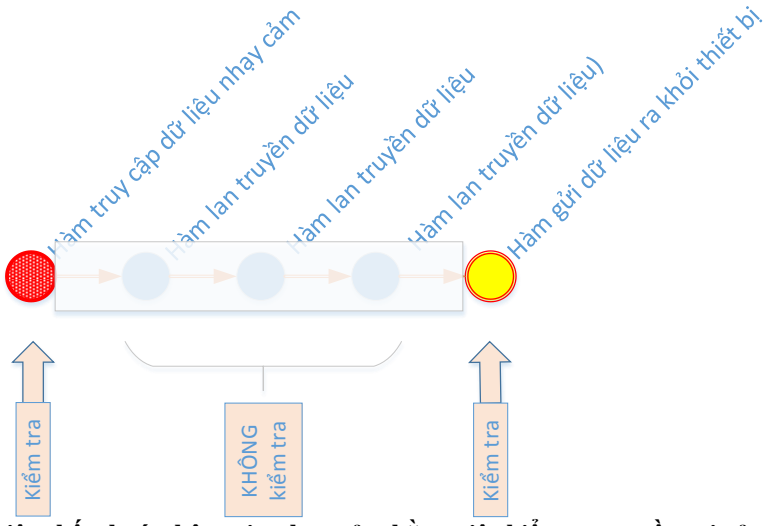


Hình 5. Cấu trúc hệ điều hành Android

Trong các nghiên cứu phân tích ứng dụng Android có thể được chia thành hai nhóm. Nhóm thứ nhất là không dựa vào việc phân tích luồng dữ liệu nhạy cảm mà dựa vào các dấu hiệu liên quan đến việc gây thất thoát thông tin nhạy cảm như sự xuất hiện hàm truy cập dữ liệu nhạy cảm, hàm gửi dữ liệu ra khỏi thiết bị [8, 9]. Nhóm thứ hai tập trung vào việc phân tích luồng

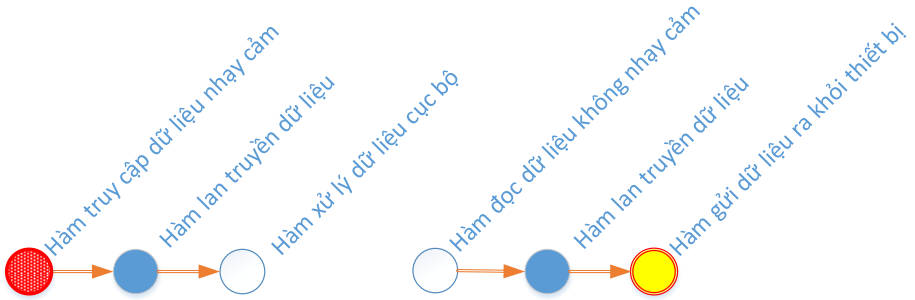
dữ liệu nhạy cảm, tức là kiểm tra chính xác dữ liệu nhạy cảm có thật sự bị gửi ra khỏi thiết bị hay không.

Với hướng nghiên cứu thứ nhất, họ tập trung vào việc phát hiện sự có mặt cùng lúc của các hàm truy cập dữ liệu nhạy cảm và hàm gửi dữ liệu ra khỏi thiết bị (Hình 6). Với hướng tiếp cận này, chúng ta có thể kiểm tra sự xuất hiện của các quyền tương ứng với *nguồn nhạy cảm* và *đích nguy hiểm*. Kirin [8] là một nghiên cứu điển hình cho hướng nghiên cứu này. Kirin kiểm tra sự xuất hiện đồng thời các quyền hạn liên quan đến *nguồn nhạy cảm* và *đích nguy hiểm* để phát hiện khả năng gây thất thoát thông tin nhạy cảm. Trong khi đó, MAD-API [9] kiểm tra sự xuất hiện của *nguồn nhạy cảm* và *đích nguy hiểm* trong các ứng dụng.



Hình 6. Phát hiện thất thoát thông tin nhạy cảm bằng việc kiểm tra sự tồn tại của *Nguồn nhạy cảm* và *Đích nguy hiểm*

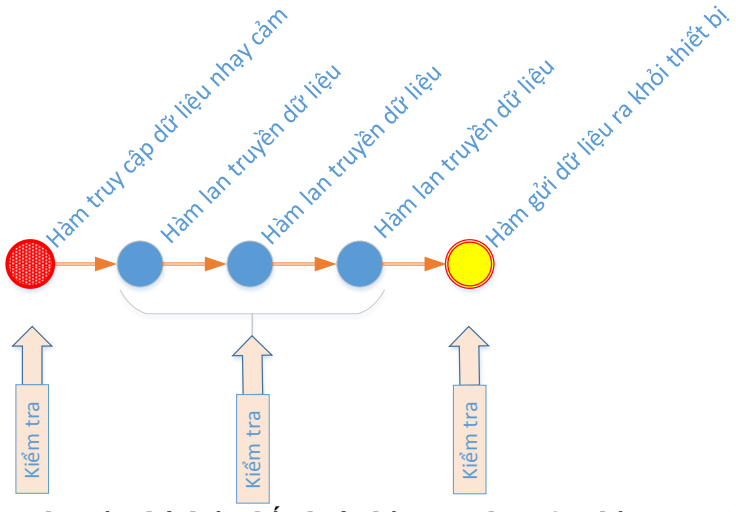
Việc không biết khả năng lan truyền dữ liệu từ *nguồn nhạy cảm* đến *đích nguy hiểm* sẽ làm giảm độ chính xác trong việc phát hiện thất thoát thông tin nhạy cảm của các nghiên cứu theo hướng này. Vì sự xuất hiện hàm truy cập dữ liệu nhạy cảm và hàm gửi dữ liệu ra khỏi thiết bị có thể là trường hợp của việc gây thất thoát dữ liệu nhạy cảm cũng có thể là trường hợp không gây thất thoát dữ liệu nhạy cảm. Ví dụ trong Hình 7, mặc dù tồn tại cả *nguồn nhạy cảm* và *đích nguy hiểm*, tuy nhiên đây không phải là trường hợp gây thất thoát thông tin nhạy cảm. Vì thông tin được gửi ra khỏi thiết bị là dữ liệu được lan truyền từ hàm đọc dữ liệu không nhạy cảm.



Hình 7. Minh họa cho trường hợp tồn tại *Nguồn nhạy cảm* và *Đích nguy hiểm* nhưng không gây thất thoát thông tin nhạy cảm

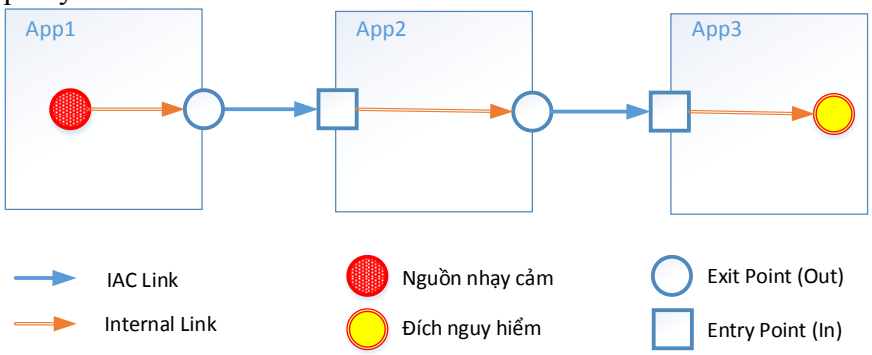
Như vậy, phân tích hành trình lan truyền dữ liệu từ *nguồn nhạy cảm* đến *đích nguy hiểm* (tức luồng dữ liệu nhạy cảm theo *Định nghĩa 6* và *Định nghĩa 7*) là điều cần thiết để phát hiện chính xác các trường hợp gây thất thoát thông tin nhạy cảm ra khỏi thiết bị (Hình 8). Đặc điểm chính của kỹ thuật này là xác định chuỗi các hàm lan truyền dữ liệu từ *nguồn nhạy cảm* đến *đích nguy hiểm* để phát hiện sự thất thoát thông tin nhạy cảm thay vì xác định sự tồn tại của *nguồn nhạy cảm* và *đích nguy hiểm*. Có nhiều nghiên cứu liên quan sử dụng hướng tiếp cận này [10-27]. FlowDroid [1] là công cụ mã nguồn mở được sử dụng phổ biến (1130 trích dẫn,

đến ngày 30/3/2019) trong các nghiên cứu liên quan đến việc phân tích luồng dữ liệu trong các ứng dụng Android. Tuy nhiên, FlowDroid chỉ hỗ trợ phân tích trong từng thành phần của ứng dụng Android, do đó cần phải cải tiến nó để hỗ trợ cho việc phân tích toàn bộ ứng dụng Android và thậm chí phân tích trên liên ứng dụng.



Hình 8. Minh họa cho việc phát hiện thất thoát thông tin nhạy cảm thông qua việc phân tích luồng dữ liệu

Phân tích luồng dữ liệu trong các ứng dụng Android có thể thực hiện theo hướng tiếp cận phân tích trên ứng dụng đơn hoặc phân tích trên liên ứng dụng. Phát hiện thất thoát thông tin nhạy cảm trong Android bằng cách phân tích luồng dữ liệu trong ứng dụng đơn sẽ gặp hạn chế trong việc phát hiện các luồng dữ liệu có hàm truy cập dữ liệu nhạy cảm và hàm gửi dữ liệu nhạy cảm ra khỏi thiết bị nằm trên hai ứng dụng khác nhau. Vì lúc đó, luồng dữ liệu được xác định trong từng ứng dụng chưa thể mang đặc điểm của một luồng dữ liệu nhạy cảm. Ví dụ trong Hình 9 thể hiện một trường hợp luồng dữ liệu gây thất thoát thông tin nhạy cảm trải dài qua ba ứng dụng. Cụ thể *nguồn nhạy cảm* và *đích nguy hiểm* nằm ở hai ứng dụng khác nhau (*App1* và *App2*). Hướng tiếp cận phân tích luồng dữ liệu trên từng ứng dụng đơn sẽ không phát hiện trường hợp này.



Hình 9. Minh họa trường hợp luồng dữ liệu nhạy cảm liên ứng dụng

Như vậy, việc phân tích luồng dữ liệu trên liên ứng dụng là cần thiết để tăng độ chính xác trong việc phát hiện thất thoát thông tin nhạy cảm. Có nhiều nghiên cứu theo hướng tiếp cận phân tích luồng dữ liệu liên ứng dụng. Trong đó, điển hình là DidFail [25], ApkCombiner [28], Amandroid [10], JITANA [27].

ApkCombiner [28] được Li và đồng sự phát triển từ IccTA [29] để phát hiện rò rỉ dữ liệu nhạy cảm qua nhiều ứng dụng. Tuy nhiên, ApkCombiner thực hiện phân tích hai ứng dụng bằng cách gom hai tập tin APK của hai ứng dụng thành một tập tin APK mới từ đó phân tích tập tin APK mới này. Phương pháp này tốn nhiều thời gian nếu như số lượng tập tin APK cần phân tích lớn. Vì lúc đó, chi phí cho việc gọp các tập tin APK lớn, không tận dụng được các kết quả phân tích trước đó.

DidFail [25] được đề xuất bởi William Klieber và đồng sự để phân tích luồng dữ liệu qua nhiều ứng dụng. Trong nghiên cứu này, họ sử dụng kỹ thuật phân tích tĩnh để phân tích các luồng dữ liệu trong từng ứng dụng và sử dụng các thông tin của Implicit Intent để phát hiện sự kết hợp của các luồng dữ liệu trong các ứng dụng đơn trong việc tạo ra các luồng dữ liệu liên ứng dụng. Đây được coi là nghiên cứu tiên phong trong việc phát hiện thất thoát dữ liệu nhạy cảm qua liên ứng dụng. Tuy nhiên, DidFail chỉ phân tích thành phần Activity và chỉ sử dụng thông tin Implicit Intent trong việc phân tích các luồng dữ liệu liên ứng dụng. Điều này khiến DidFail gặp hạn chế khi phân tích các tình huống gây thất thoát thông tin nhạy cảm qua liên ứng dụng sử dụng các thành phần ứng dụng Android khác như Content Provider, Broadcast Receiver, Service hay thông qua các kênh giao tiếp khác như tập tin chia sẻ. Một hạn chế nữa của đề xuất này là không phân tích Explicit Intent. Trong khi đó theo [30], 71.22% các Intent sử dụng trong các ứng dụng Android là Explicit Intent. Như vậy cần thiết phải phân tích loại Intent này trong việc phát hiện thất thoát thông tin nhạy cảm qua nhiều ứng dụng Android. Bên cạnh đó, trong quá trình phân tích các giao tiếp liên ứng dụng, nghiên cứu này không đề cập đến yếu tố quyền hạn cần thiết để thực thi chúng. Điều này dẫn đến kết quả phân tích của nghiên cứu này chứa tỷ lệ *False Positive* cao, vì có nhiều trường hợp các giao tiếp liên ứng dụng không thể xảy ra, tuy nhiên hệ thống vẫn cho rằng chúng là thành phần của một luồng dữ liệu liên ứng dụng. Bên cạnh đó, DidFail sử dụng kỹ thuật phân tích tĩnh nên sẽ gặp hạn chế trong việc phân tích các ứng dụng sử dụng kỹ thuật làm rối mã nguồn và dữ liệu phát sinh động khi thực thi ứng dụng.

Cũng giống như DidFail, Amandroid [10] được đề xuất bởi Wei và đồng sự để phân tích luồng dữ liệu trong các ứng dụng Android bằng kỹ thuật phân tích tĩnh. Amandroid cũng phân tích Intent để ánh xạ các luồng dữ liệu trong từng ứng dụng đơn. Tuy nhiên, Amandroid phân tích cả *Implicit Intent* và *Explicit Intent*. Trong nghiên cứu này, họ cũng không phân tích yếu tố quyền hạn trong việc xét các giao tiếp liên ứng dụng.

Các nghiên cứu liên quan đến việc kết hợp phân tích tĩnh và phân tích động hiện tại chủ yếu áp dụng cho trường hợp phân tích ứng dụng đơn [31-34]. Trong hướng nghiên cứu này, kết quả của quá trình phân tích tĩnh sẽ giúp định hướng quá trình phân tích động để quá trình phân tích diễn ra nhanh hơn.

AUNTIEDROID [33] kết hợp phân tích tĩnh và phân tích động. Trong nghiên cứu này, họ dùng phân tích tĩnh để xác định các execution path đến các lời gọi hàm nguy hiểm và kiểm chứng bằng phân tích động. Tuy nhiên, họ không phân tích luồng dữ liệu mà chỉ phát hiện sự hiện diện của các lời gọi hàm API nguy hiểm. Bên cạnh đó, họ không phân tích liên ứng dụng.

Cũng dùng kỹ thuật phân tích kết hợp, [31] dùng kỹ thuật phân tích tĩnh và động để phân tích luồng dữ liệu trong ứng dụng đơn. Họ dùng FlowDroid, kèm theo Log module để ánh xạ các giao tiếp liên thành phần để xác định các luồng dữ liệu trong ứng dụng đơn. Cũng giống nhiều nghiên cứu khác, họ không phân tích trên liên ứng dụng.

Như vậy, theo các thống kê trên, các nghiên cứu theo hướng phát hiện thất thoát thông tin nhạy cảm trong ứng dụng Android cần phải thực hiện phân tích trên liên ứng dụng thay vì phân tích từng ứng dụng đơn.

Bên cạnh số lượng lớn các nghiên cứu tập trung phân tích các ứng dụng Android. Các nghiên cứu khác tập trung phân tích firmware Android trong đó có thể bao gồm thành phần ứng dụng cài sẵn. Các nghiên cứu liên quan đến việc phân tích firmware Android tập trung vào các hướng tiếp cận khác nhau như: quét mã độc trong các ứng dụng cài sẵn [35], phân tích thất thoát dữ liệu nhạy cảm trong các ứng dụng cài sẵn [36], phân tích sự mâu thuẫn giữa các thông tin cấu hình của hệ điều hành Android cần phân tích với mô tả của nhà sản xuất [37], phát hiện sự khác biệt giữa danh sách lời gọi hàm của hệ điều hành Android cần phân tích và hệ điều hành Android chuẩn (AOSP- Android Open Source Project) [38], EdgeMiner [39] kết hợp thông tin trong Application Framework để giải quyết các trường hợp luồng dữ liệu bị gián đoạn trong các ứng dụng ở tầng Applications.

Zheng và đồng sự đề xuất DroidRay [35] để phân tích nguy cơ bảo mật trong các firmware Android bằng cách quét mã độc trong các ứng dụng cài sẵn, kiểm tra lỗ hổng bảo mật liên quan đến việc sử dụng khóa mặc định trong việc ký các ứng dụng lúc biên dịch hệ điều hành, kiểm

tra bất thường trong tập tin */etc/hosts*, kiểm tra các luật bất thường trong dịch vụ *iptables*. Tuy nhiên, trong nghiên cứu này, họ không phân tích nguy cơ gây thất thoát thông tin nhạy cảm qua các ứng dụng, cũng như mức Application Framework.

Trong khi đó, Axplorer [40] được đề xuất bởi Michael Backes và đồng sự phân tích thành phần Application Framework trong các firmware Android. Cụ thể họ phân tích các hàm truy cập dữ liệu nhạy cảm trong thành phần này. Tuy nhiên, mục tiêu của nghiên cứu này là ánh xạ quyền hạn của Android và các hàm API trong Application Framework thay vì phát hiện các luồng dữ liệu nhạy cảm trong thành phần này.

Một nghiên cứu khác là DroidDiff [37] được đề xuất bởi Aafe và đồng sự để đánh giá nguy cơ bảo mật trên các firmware Android. Trong nghiên cứu này, họ đề xuất phương pháp phân tích các cấu hình trong firmware Android để phát hiện sự không nhất quán của chúng với tài liệu mô tả của nhà sản xuất và theo hiểu biết của người phân tích. Trong nghiên cứu này, họ không phân tích luồng thông tin nhạy cảm trong các ứng dụng cài sẵn và thành phần Application Framework. Điều này cũng gây ra hạn chế giống như DroidRay là không thể phát hiện các luồng dữ liệu nhạy cảm qua nhiều ứng dụng và luồng dữ liệu nhạy cảm trong Application Framework.

Nếu như FlowDroid [1] thất bại trong việc phân tích các luồng dữ liệu bị gián đoạn bởi các hàm call back thì EdgeMiner [41] phân tích các hàm trong Application Framework để gắn kết các luồng dữ liệu lại với nhau. Ở nghiên cứu này, họ không theo dõi dữ liệu nhạy cảm trong Applicaton Framework. Họ phân tích Application Framework để hỗ trợ phân tích các ứng dụng đơn chứ không phân tích rò rỉ dữ liệu trong Application Framework.

Zhou và đồng sự đề xuất ADDICTED [38] để phân tích các firmware Android theo hướng so sánh sự khác biệt giữa danh sách lời gọi hàm mức Linux Kernel của firmware Android cần phân tích và firmware Android chuẩn (AOSP- Android Open Source Project). Trong nghiên cứu này, họ dùng phương pháp phân tích động để ghi lại các lời gọi hàm hệ thống trong AOSP và lời gọi hàm hệ thống trong firmware Android cần phân tích. Bước tiếp theo, họ phân tích sự khác biệt giữa chúng để đưa ra kết luận phân tích. Hướng tiếp cận này gặp hạn chế trong việc phát hiện luồng dữ liệu qua nhiều ứng dụng và luồng dữ liệu nhạy cảm trong thành phần Application Framework.

Cũng so sánh với các AOSP, SEFA [42] phân tích các firmware Android bằng cách phân tích các ứng dụng cài sẵn. Cụ thể họ phân tích nguồn gốc của các ứng dụng, quyền hạn của các ứng dụng và so sánh với các firmware chuẩn của AOSP. Họ không phân tích luồng dữ liệu nhạy cảm qua nhiều ứng dụng cài sẵn cũng như thành phần Application Framework.

Tóm lại, các nghiên cứu theo hướng phân tích firmware Android như các nghiên cứu kể trên đều gặp hạn chế trong việc phát hiện các luồng dữ liệu nhạy cảm qua liên ứng dụng và các luồng dữ liệu trong thành phần Application Framework. Như vậy, khi phân tích nguy cơ gây thất thoát thông tin nhạy cảm trong hệ điều hành Android cần phân tích cả thành phần ứng dụng cài sẵn, và Application Framework. Trong đó, thành phần ứng dụng cài sẵn cần phải thực hiện phân tích liên ứng dụng thay vì phân tích ứng dụng đơn.

Luận án tập trung phân tích firmware Android. Tức là dữ liệu đầu vào là firmware Android thay vì các ứng dụng Android. Cụ thể, hướng tiếp cận của luận án là phân tích hai thành phần chính của firmware Android là các ứng dụng cài sẵn (lớp Applications - Hình 5) và Application Framework (lớp Application Framework - Hình 5). Trong đó, phân tích các ứng dụng Android theo hướng tiếp cận phân tích luồng dữ liệu nhạy cảm liên ứng dụng; phân tích thành phần Application Framework theo hướng tiếp cận phân tích luồng dữ liệu nhạy cảm. Với hướng tiếp cận này, mục tiêu của luận án là rút trích thành công các thành phần chính của firmware Android, giảm tỷ lệ False Positive và False Negative trong việc phát hiện luồng thông tin nhạy cảm qua nhiều ứng dụng, giảm tỷ lệ False Positive trong việc phát hiện luồng thông tin nhạy cảm trong thành phần Application Framework.

2.3 Các thách thức hiện tại

Để phân tích các thành phần của hệ điều hành Android, việc rút trích các thành phần này cần phải có độ chính xác cao. Hiện tại có nhiều hãng điện thoại khác nhau với các tùy chỉnh

cấu hình khác nhau, do đó việc xây dựng giải pháp để rút trích được các thành phần hệ điều hành từ các hãng khác nhau cũng là một thách thức. Mặc dù thách thức này mang tính kỹ thuật là chính.

Một khi đã rút trích được các thành phần trong hệ điều hành Android, thách thức kế tiếp là phân tích nguy cơ gây thất thoát thông tin nhạy cảm trong các thành phần này. Phân tích từng thành phần riêng biệt hay phân tích kết hợp các thành phần này cũng là thách thức lớn trong việc đề xuất các giải pháp phân tích liên quan.

Xu thế tấn công leo thang quyền hạn thông qua việc kết hợp nhiều ứng dụng trong một đợt tấn công đòi hỏi các bộ phân tích phải có thông tin của nhiều ứng dụng trong việc phân tích. Điều này là thách thức cho các giải pháp phân tích trên ứng dụng đơn như IccTA [29], SmartDroid [32], FlowDroid [1], Ripple [26].

Một khi đã mở rộng đối tượng phân tích từ ứng dụng đơn sang liên ứng dụng, bộ phân tích phải có khả năng đáp ứng được yêu cầu phân tích cùng lúc chuỗi nhiều ứng dụng. Điều này đòi hỏi giải pháp phân tích trên liên ứng dụng phải sử dụng mô hình toán học phù hợp trong việc gắn kết thông tin trên liên ứng dụng, phân tích thông tin trên liên ứng dụng. Các nghiên cứu hiện tại như Mr-Droid [43] chỉ phân tích từng cặp hai ứng dụng; DidFail [25] chỉ phân tích các ứng dụng trong bộ dữ liệu thử nghiệm DroidBench [44] với chuỗi ba ứng dụng.

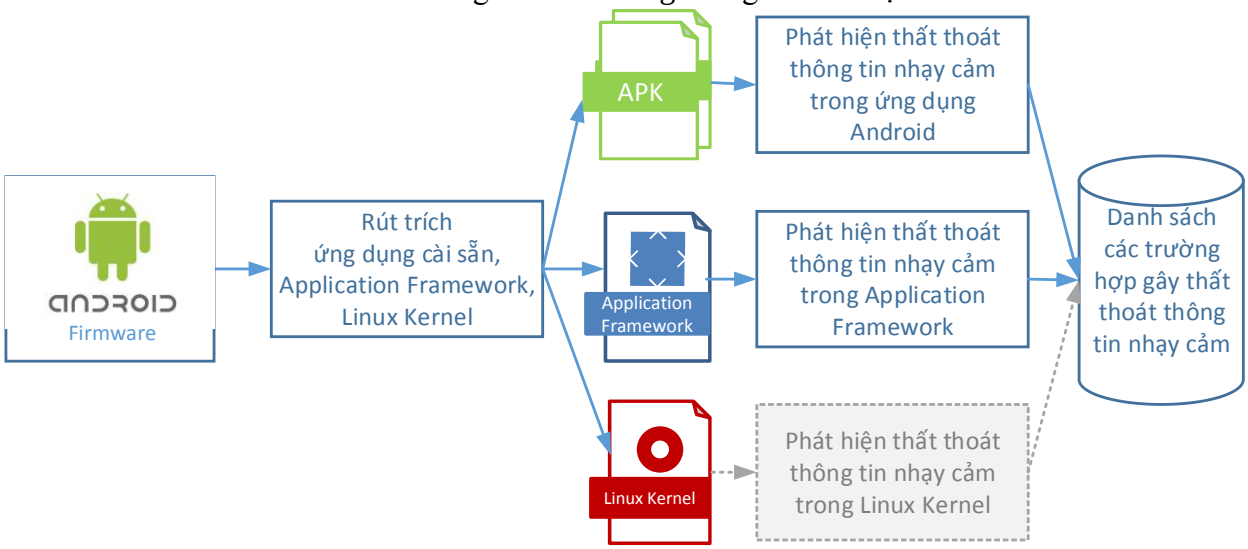
Một khi đối tượng phân tích là liên ứng dụng thì việc phân tích chính xác các giao tiếp liên ứng dụng sẽ giúp ích trong việc tăng độ chính xác của các giải pháp đề xuất. Phân tích chính xác các giao tiếp liên ứng dụng bao gồm: phân tích được nhiều loại giao tiếp liên ứng dụng, xác định chính xác sự tồn tại của các giao tiếp liên ứng dụng. Các nghiên cứu hiện tại như DidFail [25] chỉ phân tích giao tiếp liên ứng dụng thông qua Implicit Intent với thành phần Activity, không kiểm tra khả năng tồn tại của các giao tiếp liên ứng dụng. Điều này dẫn đến DidFail liệt kê các giao tiếp liên ứng dụng thật sự không tồn tại, làm giảm độ chính xác. Cần mở rộng phạm vi phân tích các loại giao tiếp liên ứng dụng khác như thông qua tập tin chia sẻ, Explicit Intent, đồng thời kiểm tra sự tồn tại của các giao tiếp liên ứng dụng.

Một thách thức lớn của kỹ thuật phân tích tĩnh là gặp khó khăn trong việc phân tích các ứng dụng có mã nguồn đã được làm rối, có mã nguồn được tải lúc thực thi ứng dụng. Bên cạnh đó, kết quả phân tích giao tiếp liên ứng dụng bằng kỹ thuật phân tích tĩnh chứa nhiều giao tiếp được dự đoán mà không thể kiểm chứng chúng có tồn tại lúc thực thi ứng dụng hay không. Việc sử dụng kỹ thuật phân tích động sẽ giúp ích trong việc giải quyết các thách thức này. Tuy nhiên, việc sử dụng phân tích động như thế nào để không tương tác lên tất cả các thành phần giao diện của ứng dụng nhằm tiết kiệm thời gian phân tích là một thách thức cho các nghiên cứu hiện tại. SmartDroid là một nghiên cứu điển hình trong sử dụng phân tích tĩnh để điều hướng quá trình phân tích động nhằm tiết kiệm thời gian phân tích. Tuy nhiên nghiên cứu này chỉ làm việc trên ứng dụng đơn. Cải tiến SmartDroid để tạo giải pháp phân tích trên liên ứng dụng là động lực nghiên cứu hợp lý cho việc phân tích thất thoát thông tin nhạy cảm qua nhiều ứng dụng.

Chương 3. HƯỚNG TIẾP CẬN CỦA LUẬN ÁN

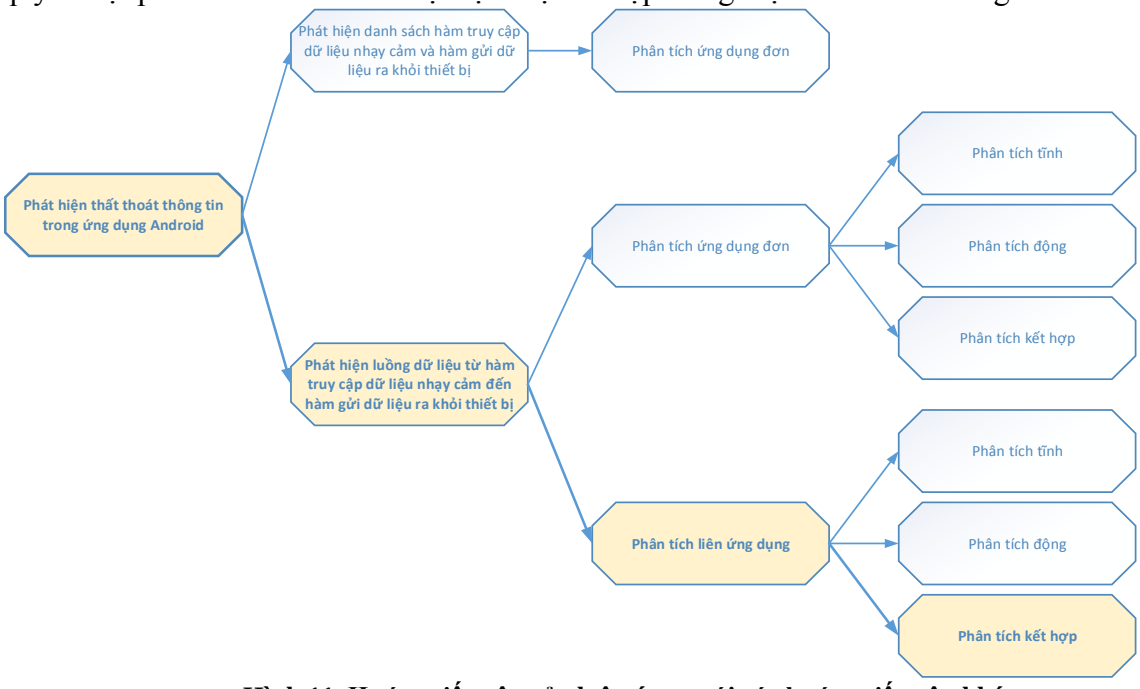
3.1 Lý do lựa chọn hướng tiếp cận của luận án

Hướng tiếp cận của luận án là phát hiện thất thoát thông tin nhạy cảm trong các thiết bị sử dụng hệ điều hành Android bằng cách phát hiện thất thoát thông tin nhạy cảm trong các thành phần quan trọng của Firmware Android. Trong luận án này, chúng tôi giới hạn phạm vi nghiên cứu ở mức phát hiện thất thoát thông tin nhạy cảm trong hai thành phần ở mức cao nhất của firmware Android là các ứng dụng cài sẵn và Application Framework. Việc phân tích các thành phần khác như Linux Kernel sẽ là bước nghiên cứu trong tương lai của luận án.



Hình 10. Hướng tiếp cận của luận án

Đối với việc phát hiện thất thoát thông tin nhạy cảm trong ứng dụng Android, luận án đi theo hướng tiếp cận phát hiện luồng dữ liệu nhạy cảm (theo *Định nghĩa 6, Định nghĩa 7* ở mục 2.1) bằng cách kết hợp kỹ thuật phân tích tĩnh và phân tích động trên liên ứng dụng để giải quyết một phần các thách thức hiện tại được đề cập trong mục 2.3 của Chương 2.



Hình 11. Hướng tiếp cận của luận án so với các hướng tiếp cận khác

Sở dĩ, luận án theo hướng tiếp cận phát hiện luồng dữ liệu nhạy cảm thay vì phát hiện danh sách hàm truy cập dữ liệu nhạy cảm là vì hướng tiếp cận này có thể giúp phát hiện chính

xác hơn các trường hợp gây thất thoát thông tin nhạy cảm, giúp giảm tỷ lệ phát hiện nhầm. Bên cạnh đó, việc phân tích liên ứng dụng thay cho phân tích ứng dụng đơn giúp giải quyết các hạn chế của các nghiên cứu hiện tại liên quan đến việc phát hiện các luồng dữ liệu liên ứng dụng trong các trường hợp tấn công cộng tác. Việc đi sâu vào việc phân tích các giao tiếp liên ứng dụng giúp ích trong việc tăng tính hiệu quả trong việc phân tích liên ứng dụng. Trong hướng nghiên cứu của luận án, chúng tôi sử dụng kết hợp kỹ thuật phân tích tĩnh và phân tích động để tăng độ chính xác của kết quả phân tích và giảm thời gian của quá trình phân tích so với việc sử dụng kỹ thuật phân tích tĩnh và động riêng biệt.

Đối với việc phát hiện thất thoát thông tin nhạy cảm trong thành phần Application Framework, luận án xem thành phần này như một ứng dụng Android đặc biệt và đề xuất phương pháp áp dụng kỹ thuật phân tích tĩnh trong ứng dụng Android cho thành phần này. Với hướng tiếp cận này, đề xuất của luận án có thể phát hiện chính xác hơn các trường hợp gây thất thoát thông tin nhạy cảm trong thành phần Application Framework.

3.2 Dữ liệu thử nghiệm

3.2.1 Bộ dữ liệu thử nghiệm tình huống gây thất thoát thông tin nhạy cảm qua liên ứng dụng

Các nghiên cứu liên quan hiện tại [10, 25, 27-29, 32, 45-60] sử dụng bộ dữ liệu thử nghiệm DroidBench [44] để đánh giá các phương pháp của họ. Tuy nhiên, bộ dữ liệu thử nghiệm này có ít mẫu thử. Hiện bộ dữ liệu này có 13 nhóm mẫu thử khác nhau. Trong đó, nhóm mẫu thử liên quan đến giao tiếp liên ứng dụng chỉ có 3 mẫu thử. Số mẫu thử này ít vì hướng nghiên cứu phân tích liên ứng dụng đang là hướng nghiên cứu mới. Bảng 1 mô tả cấu trúc của bộ dữ liệu thử nghiệm DroidBench dataset.

Bảng 1. Cấu trúc của DroidBench dataset

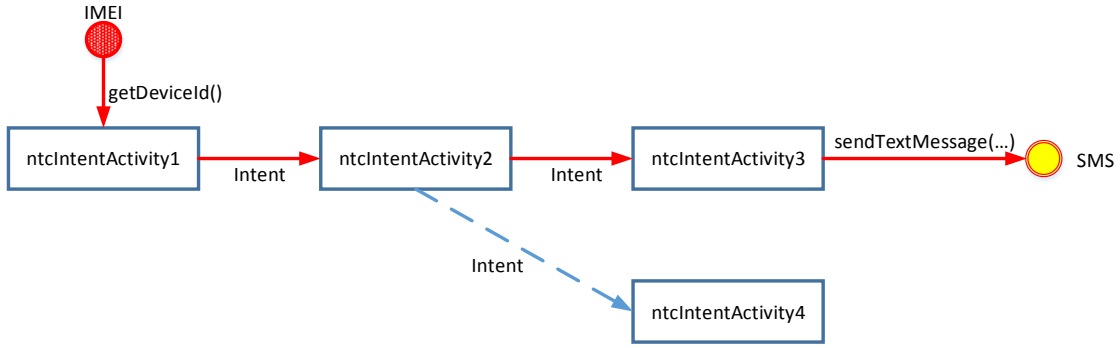
Loại mẫu	Số mẫu	Loại mẫu	Số mẫu
Alias	1	ImplicitFlows	4
AndroidSpecific	12	**InterAppCommunication**	**3**
ArraysAndLists	7	InterComponentCommunication	18
CallBacks	15	Lifecycle	17
EmulatorDetection	3	Reflection	4
FieldAndObjectSensitive	7	Threading	5
GeneralJava	23		

Để tăng độ tin cậy cho kết quả thử nghiệm trong nghiên cứu này cũng như đóng góp thêm các mẫu thử cho DroidBench dataset, chúng tôi viết thêm 312 ứng dụng để thể hiện 37 tình huống tương ứng với 45 luồng dữ liệu nhạy cảm khác nhau. Các tình huống này sử dụng 4 loại thành phần của Android là Activity, Broadcast Receiver, Service, Content Provider và một loại giao tiếp khác là tập tin chia sẻ (Shared file). Bên cạnh đó, chúng tôi xây dựng 4 tình huống sử dụng việc phát sinh khả năng gây rò rỉ thông tin khi thực thi ứng dụng (Dynamic Content) để hỗ trợ trong việc phân tích động. Ngoài ra, trong loại tình huống Shared file chúng tôi xây dựng một kịch bản rò rỉ thông tin nhạy cảm qua 200 ứng dụng (tình huống 30 trong Bảng 5) để hỗ trợ trong việc kiểm tra khả năng phân tích chuỗi nhiều ứng dụng của hệ thống đề xuất. Ngoài ra, chúng tôi xây dựng hai kịch bản có luồng dữ liệu đi qua liên lớp thành phần Inter Layer (thành phần ứng dụng và application framework). Dữ liệu thử nghiệm này được gọi là *uitIACdroid-Bench*. Bảng 2 mô tả chi tiết cấu trúc của *uitIACDroid-Bench*. Nội dung chi tiết của bộ dữ liệu thử nghiệm được trình bày trong Phụ lục A. Trong quá trình thực hiện luận án, chúng tôi đã gửi bộ dữ liệu thử nghiệm này cho 8 nhóm nghiên cứu mạnh liên quan trên thế giới để nhờ họ đánh giá. Chúng tôi nhận được các đánh giá tích cực từ họ. Chi tiết các đánh giá từ các nhóm nghiên cứu được trình bày trong Phụ lục C. Điều này chứng tỏ bộ dữ liệu thử nghiệm *uitIACDroid-Bench* được sự tán đồng cao từ cộng đồng nghiên cứu. Bên cạnh đó, một phần nội dung này được trình bày tại hội nghị ICCAI2019 [CB8].

Bảng 2. Cấu trúc của uitIACDroid-Bench

Loại tình huống	Số tình huống	Số mẫu thử
Activity	6	23
Service	6	18
Broadcast Receiver	6	18
Shared file	10	224
Content provider	3	6
Dynamic Content	4	10
Inter Layer	2	4

Hình 12 mô tả một luồng dữ liệu nhạy cảm của loại tình huống sử dụng Activity. Trong ví dụ này, các đường mũi tên nét liền thể hiện đường đi của dữ liệu nhạy cảm từ ứng dụng ntcIntentActivity1 qua ntcIntentActivity2 trước khi qua ntcIntentActivity3 để được gửi ra ngoài thông qua SMS.



Hình 12. Một tình huống trong bộ dữ liệu uitIACDroid-Bench

Với các tình huống đã biết trước trong bộ dữ liệu thử nghiệm này giúp ích trong việc kiểm tra tính đúng đắn của hệ thống đề xuất trong việc phát hiện luồng thông tin nhạy cảm qua các loại component khác nhau của ứng dụng Android. Mặc dù, các tình huống này chưa bao hàm tất cả các tình huống trong thực tế, tuy nhiên luận án tập trung phủ hết các loại component khác nhau của Android thay vì tất cả các tình huống.

3.2.2 Bộ dữ liệu thử nghiệm cho firmware Android tùy biến

Trong nghiên cứu này, chúng tôi đề xuất bộ dữ liệu thử nghiệm với 10 ROM tùy biến với các tình huống được mô tả trong Bảng 3. Bộ dữ liệu này được đặt tên là uitCustomROM-Bench. Để tạo các firmware Android tùy biến cho bộ dữ liệu thử nghiệm này, chúng tôi chèn vào thành phần Application Framework của mã nguồn firmware Android chuẩn từ [61] để tạo các firmware Android có khả năng làm rò rỉ dữ liệu nhạy cảm theo các kịch bản khác nhau. Việc biết rõ các kịch bản gây rò rỉ dữ liệu nhạy cảm giúp đánh giá khả năng làm việc của các hệ thống đề xuất.

Bảng 3. Thông tin mô tả 10 mẫu thử nghiệm trong bộ dataset uitCustomROM-Bench đề xuất

STT	Loại nguồn nhạy cảm	Loại đích nguy hiểm	Ghi chú
1	Contact list	Socket	Đọc danh bạ và gởi đến một dịch vụ trên Internet thông qua một socket.
2	Contact list	SMS	Đọc danh bạ và gởi qua tin nhắn SMS
3	GPS+IMEI	Socket	Lấy vị trí, mã thiết bị rồi gởi đến một dịch vụ trên Internet thông qua một socket.
4	GPS+IMEI	SMS	Lấy vị trí, mã thiết bị rồi gởi qua tin nhắn SMS.
5	IMEI	Socket	Lấy mã thiết bị rồi gởi đến một dịch vụ trên Internet qua một socket.
6	IMEI	SMS	Lấy mã thiết bị rồi gởi qua tin nhắn SMS.

7	Call log	Socket	Lấy lịch sử cuộc gọi rồi gởi đến một dịch vụ trên Internet qua một socket.
8	Call log	SMS	Lấy lịch sử cuộc gọi rồi gửi qua tin nhắn SMS.
9	Contact list	Internet	Lấy danh bạ rồi gởi lên Internet.
10	Call log	Internet	Lấy lịch sử cuộc gọi rồi gởi lên Internet.

Bên cạnh đó, chúng tôi tải 290 firmware Android từ các nguồn chia sẻ trên Internet để đánh giá khả năng phân tích các firmware Android tùy biến ngoài thế giới thực của phương pháp đề xuất.

3.3 Các đề xuất của luận án

Ba đóng góp chính của luận án bao gồm:

- Đề xuất phương pháp phát hiện thất thoát thông tin nhạy cảm trong các ứng dụng Android với độ chính xác được cải thiện. Cụ thể phương pháp đề xuất cho phép mở rộng các loại giao tiếp liên ứng dụng, kiểm tra khả năng tồn tại của các giao tiếp liên ứng dụng, tiết kiệm thời gian trong quá trình phân tích động.
- Đề xuất phương pháp áp dụng kỹ thuật phân tích luồng dữ liệu trong ứng dụng Android để phát hiện thất thoát thông tin nhạy cảm trong thành phần Application Framework trong firmware Android.
- Phát triển các công cụ phục vụ cho quá trình thực hiện hai đóng góp trên của luận án. Mặc dù, đóng góp thứ ba mang tính kỹ thuật là chủ yếu, tuy nhiên công sức mà chúng tôi bỏ ra là đáng ghi nhận trong quá trình thực hiện luận án.

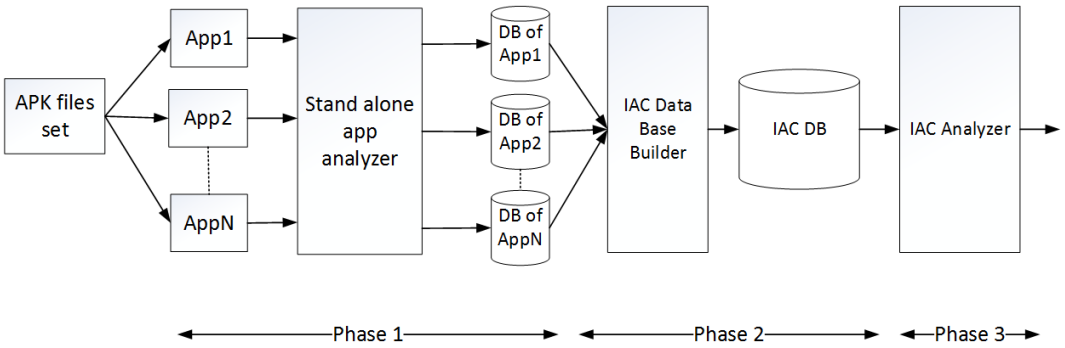
3.3.1 Đề xuất phương pháp cải thiện độ chính xác trong việc phát hiện thất thoát thông tin nhạy cảm qua liên ứng dụng

Luận án đề xuất phương pháp cải thiện độ chính xác trong việc phát hiện thất thoát thông tin nhạy cảm qua liên ứng dụng. Cụ thể phương pháp đề xuất cho phép mở rộng các loại giao tiếp liên ứng dụng, kiểm tra khả năng tồn tại của các giao tiếp liên ứng dụng, tiết kiệm thời gian trong quá trình phân tích động.

3.3.1.1 Đề xuất phương pháp cải thiện độ chính xác của việc phân tích luồng dữ liệu liên ứng dụng bằng cách phân tích mở rộng các loại giao tiếp liên ứng dụng

Ứng dụng Android có 4 loại thành phần gồm Activity, Service, Broadcast Receiver, Content Provider. Trong đó, thành phần Activity phổ biến nhất trong các ứng dụng. Theo [62], trong số 13.944 ứng dụng miễn phí phổ biến trên Google Play, thành phần Activity chiếm 77%. Các nghiên cứu hiện tại như DidFail [25] chỉ phân tích thành phần này. Đối tượng giao tiếp phổ biến giữa các thành phần ứng dụng Android là Intent. Theo [30], có đến 71.22% các Intent sử dụng trong các ứng dụng Android là Explicit Intent. Trong khi đó, DidFail lại phân tích Implicit Intent. Luận án đề xuất mở rộng các loại giao tiếp bao gồm các loại giao tiếp liên quan đến cả bốn thành phần ứng dụng Android và tập tin chia sẻ. Bên cạnh đó, luận án phân tích cả hai loại của đối tượng Intent là Implicit Intent và Explicit Intent. Một phần nội dung của đề xuất này (đặt tên là IACDroid) được trình bày tại Hội nghị ICISA 2016 [CB4].

Hệ thống đề xuất có 3 thành phần chính: *Standalone application analyzer*, *IAC DataBase Builder* và *IAC Analyzer*. Hình 13 mô tả các thành phần của IACDroid



Hình 13. Sơ đồ hệ thống IACDroid

Nếu như các hướng tiếp cận phân tích ứng dụng đơn chỉ có Phase 1 như hệ thống đề xuất, các nghiên cứu theo hướng tiếp cận phân tích liên ứng dụng sẽ thực hiện thêm Phase 2 và Phase 3 để phát hiện các luồng dữ liệu liên ứng dụng. *Stand alone app analyzer* phân tích luồng dữ liệu trong từng ứng dụng, trong khi đó *IAC Analyzer* sẽ phân tích tính liên thông giữa các luồng dữ liệu ở các ứng dụng khác nhau trong Phase 1 để tìm ra luồng dữ liệu nhạy cảm liên ứng dụng.

Với đề xuất cải tiến này, luận án cải thiện độ chính xác hơn so với công trình nghiên cứu liên quan. Cụ thể, khi thử nghiệm với bộ dữ liệu thử nghiệm uitIACDroid-Bench phương pháp đề xuất này có độ chính xác là 87,04% so với 76,19% của DidFail. Do DidFail chỉ phân tích thành phần ứng dụng Activity, do đó nghiên cứu này thất bại trong việc phân tích 24 tình huống sử dụng Service, Broadcast Receiver, Content Provider và Shared file trong bộ dữ liệu thử nghiệm uitIACDroid-Bench. Bên cạnh đó, DidFail chỉ phân tích các Intent thuộc dạng Implicit Intent do đó thất bại trong việc phân tích hai tình huống Activity sử dụng Explicit Intent.

Bảng 4. Kết quả so sánh giữa phương pháp đề xuất IACDroid và DidFail

					DidFail			IACDroid		
Data Set	**STT**	**Nhóm tình huống**	**Tình huống**	**Số luồng thực tế**	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative
DroidBench	1	Activity	SendSMS + StartActivityForResult1 + Echoer	10	10	1	0	10	1	0
uitIACDroid-Bench	2	Activity	ntcIntentActivity1 + ntcIntentActivity2 + ntcIntentActivity3 + ntcIntentActivity4	3	0	0	3	3	0	0
	3	Activity	ntcIntentActivity5 + ntcIntentActivity6 + ntcIntentActivity7 + ntcIntentActivity8	3	0	0	3	3	0	0
	4	Activity	ntcIntentActivity9 + ntcIntentActivity10 + ntcIntentActivity11 + ntcIntentActivity12	4	4	1	0	4	1	0

	5	Activity	ntcIntentActivity13 + ntcIntentActivity14 + ntcIntentActivity15	2	2	1	0	2	1	0
	6	Activity	ntcIntentActivityPermission1 + ntcIntentActivityPermission2 + ntcIntentActivityPermission3 + ntcIntentActivityPermission4	1	0	1	0	0	1	0
	7	Activity	ntcIntentActivityPermission5 + ntcIntentActivityPermission6 + ntcIntentActivityPermission7 + ntcIntentActivityPermission8	1	0	1	0	0	1	0
	8	Service	ntcService1 + ntcService2 + ntcService3	1	0	0	1	1	0	0
	9	Service	ntcService4 + ntcService5 + ntcService6	1	0	0	1	1	0	0
	10	Service	ntcService7 + ntcService8 + ntcService9 + ntcService10	1	0	0	1	1	0	0
	11	Service	ntcService11 + ntcService12 + ntcService13 + ntcService14	1	0	0	1	1	0	0
	12	Service	ntcService15 + ntcService16	1	0	0	1	1	0	0
	13	Service	ntcService17 + ntcService18	1	0	0	1	1	0	0
	14	Broadcast Receiver	ntcBroadcast1 + ntcBroadcast2 + ntcBroadcast3	1	0	0	1	1	0	0
	15	Broadcast Receiver	ntcBroadcast4 + ntcBroadcast5 + ntcBroadcast6	1	0	0	1	1	0	0
	16	Broadcast Receiver	ntcBroadcast7 + ntcBroadcast8 + ntcBroadcast9	1	0	0	1	1	0	0
	17	Broadcast Receiver	ntcBroadcast10 + ntcBroadcast11 + ntcBroadcast12	1	0	0	1	1	0	0
	18	Broadcast Receiver	ntcBroadcast13 + ntcBroadcast14 + ntcBroadcast15	1	0	0	1	1	0	0
	19	Broadcast Receiver	ntcBroadcast16 + ntcBroadcast17 + ntcBroadcast18	1	0	0	1	1	0	0
	10	Shared file	ntcWriteFile1 + ntcReadFile2 + ntcReadFile1	1	0	0	1	1	0	0
	21	Shared file	ntcWriteFile3 + ntcReadFile3 + ntcWriteFile4	1	0	0	1	1	0	0
	22	Shared file	ntcWriteFile5 + ntcReadFile5	1	0	0	1	1	0	0
	23	Shared file	ntcWriteFile6 + ntcReadFile6	1	0	0	1	1	0	0
	24	Shared file	ntcWriteFile7 + ntcReadFile7	1	0	0	1	1	0	0
	25	Shared file	ntcWriteFile8 + ntcReadFile8 + ntcReadAndWriteFile8	1	0	0	1	1	0	0

	26	Shared file	ntcWriteFile9 + ntcReadFile9 + ntcReadAndWriteFile9	1	0	0	1	1	0	0
	27	Shared file	ntcWriteFile10 + ntcReadFile10 + ntcReadAndWriteFile10	1	0	0	1	1	0	0
	28	Shared file	ntcWriteFile11 + ntcReadFile11 + ntcReadAndWriteFile11	1	0	0	1	1	0	0
	29	Shared file	ntcSharedFileChain1+ … + ntcSharedFileChain200	1	0	0	1	1	0	0
	30	Content provider	ntcContentProvider1 + ntcContentProvider2	1	0	0	1	1	0	0
	31	Content provider	ntcContentProvider3 + ntcContentProvider4	1	0	0	1	1	0	0
	32	Content provider	ntcContentProvider5 + ntcContentProvider6	1	0	0	1	1	0	0
	33	Dynamic Content	ntcDynamicContent1 + ntcDynamicContent2	1	0	0	1	0	0	1
	34	Dynamic Content	ntcDynamicContent3 + ntcDynamicContent4 + ntcDynamicContent5	1	0	0	1	0	0	1
	35	Dynamic Content	ntcDynamicContent6 + ntcDynamicContent7	1	0	0	1	0	0	1
	36	Dynamic Content	ntcDynamicContent8 + ntcDynamicContent9 + ntcDynamicContent10	1	0	0	1	0	0	1
	37	Inter Layer	ntcInterLayer1 + ntcInterLayer2	1	0	0	1	0	1	1
	38	Inter Layer	ntcInterLayer3 + ntcInterLayer4	1	0	0	1	0	1	1
Tổng		Tổng số		55	16	5	37	47	7	6
		Precision p = TP/(TP+FP) = ☑/(☑+☒)			76,19%			87,04%		
		Recall r = TP/(TP+FN) = ☑/(☑+☐)			30,19%			88,68%		
		F1 measure = 2pr/(p+r)			43,24%			87,85%		

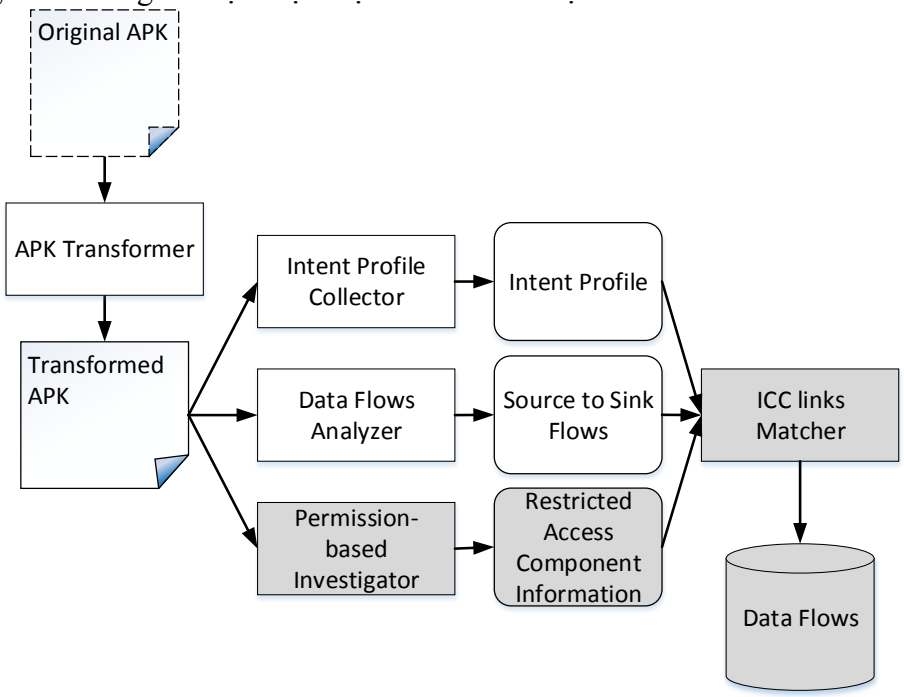
3.3.1.2 Đề xuất phương pháp cải thiện độ chính xác của việc phân tích luồng dữ liệu liên ứng dụng bằng cách bổ sung mô hình quyền hạn

Trong phần trước luận án trình bày đề xuất mở rộng các loại giao tiếp liên ứng dụng trong việc phát hiện thất thoát thông tin nhạy cảm qua chuỗi nhiều ứng dụng bằng kỹ thuật phân tích tĩnh. Tuy nhiên cũng giống một số công cụ phân tích khác như Amandroid[10], IccTA [29], DidFail [25] chúng đều không xét khả năng thực hiện thành công các giao tiếp liên thành phần, giao tiếp liên ứng dụng trong các luồng dữ liệu. Cụ thể chúng không kiểm tra các quyền hạn được cấp cho các giao tiếp liên thành phần, liên ứng dụng.

Trong nghiên cứu này, chúng tôi đề xuất một số cải tiến nhằm loại bỏ một số luồng dữ liệu không chính xác bằng cách xét các yếu tố quyền hạn khi gọi đến một thành phần ứng dụng và thuộc tính "*exported*" của Activity. Một phần nội dung của phương pháp này (đặt tên là eddLeak) được trình bày tại hội nghị ICCSN2017 [CB6].

Trong thực tế, có một số lời gọi giao tiếp đến Activity của ứng dụng khác bị từ chối do không gán thuộc tính "*exported=true*" cho Activity được gọi, hoặc do không cung cấp đúng quyền được yêu cầu cho ứng dụng nguồn. Trong các trường hợp này, không thể có đường đi giữa hai ứng dụng được. Tuy nhiên, các nghiên cứu hiện tại như IccTA[29], DidFail [25] và IACDroid [CB4] vẫn xem đây là các đường đi khả dĩ của các luồng dữ liệu.

Trong nghiên cứu này chúng tôi đề xuất một hướng tiếp cận trong việc cải tiến thuật toán ánh xạ các thành phần của giao tiếp liên ứng dụng trong việc phân tích nguy cơ thất thoát thông tin nhạy cảm trên Android bằng cách xét yếu tố quyền hạn trong việc thực hiện các lời gọi hàm liên thành phần (ICC) và liên ứng dụng (IAC). Hệ thống đề xuất có 5 thành phần chính ***APK Transformer, Intent Profile Collector, Data Flows Analyzer, Permission-based Investigator, ICC links***. Trong đó hai mô-đun ***Permission-based Investigator*** và ***ICC links*** là hai mô-đun được chúng tôi bổ sung để thực hiện mục tiêu cải tiến độ chính xác.



Hình 14. Sơ đồ hệ thống của phương pháp eddLeak

Nghiên cứu này, chúng tôi sử dụng *DroidBench* và *uitIACDroid-Bench* để kiểm tra khả năng phân tích các đề xuất trong luận án IACDroid, eddLeak với công trình liên quan trực tiếp là DidFail [25]. Bảng 5 mô tả kết quả phân tích của các công trình này, khi dùng nhóm mẫu thử *InterAppCommunication* trong *DroidBench* và 312 mẫu thử trong *uitIACDroid-Bench*. Kết quả phân tích cho thấy hệ thống đề xuất phát hiện chính xác các tình huống rò rỉ thông tin biết trước trong bộ dữ liệu thử nghiệm đề xuất trừ các tình huống thuộc loại dữ liệu tự phát sinh lúc chạy chương trình và luồng dữ liệu liên lớp thành phần (tình huốn 37 và 38). Bên cạnh đó, các nghiên cứu này vẫn còn phát hiện nhầm (False Positive) 5 trường hợp. Cụ thể trong các tình huống 1,4,5 và 6 chúng phát hiện nhầm 5 luồng dữ liệu do kỹ thuật ánh xạ Intent và Intent Filter trong các loại Explicit Intent. Hạn chế này có thể được giải quyết nếu chúng ta sử dụng kỹ thuật phân tích động thay cho phân tích tĩnh.

Bảng 5. Kết quả phân tích của DidFail, IACDroid và eddLeak

					DidFail			IACDroid			eddLeak		
Data Set	**STT**	**Nhóm tình huống**	**Tình huống**	**Số luồng thực tế**	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative

DroidBench	1	Activity	SendSMS + StartActivityForResult1 + Echoer	10	10	1	0	10	1	0	10	1	0
uitIACDroid-Bench	2	Activity	ntcIntentActivity1 + ntcIntentActivity2 + ntcIntentActivity3 + ntcIntentActivity4	3	0	0	3	3	0	0	3	0	0
	3	Activity	ntcIntentActivity5 + ntcIntentActivity6 + ntcIntentActivity7 + ntcIntentActivity8	3	0	0	3	3	0	0	3	0	0
	4	Activity	ntcIntentActivity9 + ntcIntentActivity10 + ntcIntentActivity11 + ntcIntentActivity12	4	4	1	0	4	1	0	4	1	0
	5	Activity	ntcIntentActivity13 + ntcIntentActivity14 + ntcIntentActivity15	2	2	1	0	2	1	0	2	1	0
	6	Activity	ntcIntentActivityPermission1 + ntcIntentActivityPermission2 + ntcIntentActivityPermission3 + ntcIntentActivityPermission4	1	0	1	0	0	1	0	1	0	0
	7	Activity	ntcIntentActivityPermission5 + ntcIntentActivityPermission6 + ntcIntentActivityPermission7 + ntcIntentActivityPermission8	1	0	1	0	0	1	0	1	0	0
	8	Service	ntcService1 + ntcService2 + ntcService3	1	0	0	1	1	0	0	1	0	0
	9	Service	ntcService4 + ntcService5 + ntcService6	1	0	0	1	1	0	0	1	0	0
	10	Service	ntcService7 + ntcService8 + ntcService9 + ntcService10	1	0	0	1	1	0	0	1	0	0
	11	Service	ntcService11 + ntcService12 + ntcService13 + ntcService14	1	0	0	1	1	0	0	1	0	0
	12	Service	ntcService15 + ntcService16	1	0	0	1	1	0	0	1	0	0
	13	Service	ntcService17 + ntcService18	1	0	0	1	1	0	0	1	0	0
	14	Broadcast Receiver	ntcBroadcast1 + ntcBroadcast2 + ntcBroadcast3	1	0	0	1	1	0	0	1	0	0
	15	Broadcast Receiver	ntcBroadcast4 + ntcBroadcast5 + ntcBroadcast6	1	0	0	1	1	0	0	1	0	0
	16	Broadcast Receiver	ntcBroadcast7 + ntcBroadcast8 + ntcBroadcast9	1	0	0	1	1	0	0	1	0	0
	17	Broadcast Receiver	ntcBroadcast10 + ntcBroadcast11 + ntcBroadcast12	1	0	0	1	1	0	0	1	0	0
	18	Broadcas	ntcBroadcast13 +	1	0	0	1	1	0	0	1	0	0

		t Receiver	ntcBroadcast14 + ntcBroadcast15										
	19	Broadcast Receiver	ntcBroadcast16 + ntcBroadcast17 + ntcBroadcast18	1	0	0	1	1	0	0	1	0	0
	10	Shared file	ntcWriteFile1 + ntcReadFile2 + ntcReadFile1	1	0	0	1	1	0	0	1	0	0
	21	Shared file	ntcWriteFile3 + ntcReadFile3 + ntcWriteFile4	1	0	0	1	1	0	0	1	0	0
	22	Shared file	ntcWriteFile5 + ntcReadFile5	1	0	0	1	1	0	0	1	0	0
	23	Shared file	ntcWriteFile6 + ntcReadFile6	1	0	0	1	1	0	0	1	0	0
	24	Shared file	ntcWriteFile7 + ntcReadFile7	1	0	0	1	1	0	0	1	0	0
	25	Shared file	ntcWriteFile8 + ntcReadFile8 + ntcReadAndWriteFile8	1	0	0	1	1	0	0	1	0	0
	26	Shared file	ntcWriteFile9 + ntcReadFile9 + ntcReadAndWriteFile9	1	0	0	1	1	0	0	1	0	0
	27	Shared file	ntcWriteFile10 + ntcReadFile10 + ntcReadAndWriteFile10	1	0	0	1	1	0	0	1	0	0
	28	Shared file	ntcWriteFile11 + ntcReadFile11 + ntcReadAndWriteFile11	1	0	0	1	1	0	0	1	0	0
	29	Shared file	ntcSharedFileChain1+ … + ntcSharedFileChain200	1	0	0	1	1	0	0	1	0	0
	30	Content provider	ntcContentProvider1 + ntcContentProvider2	1	0	0	1	1	0	0	1	0	0
	31	Content provider	ntcContentProvider3 + ntcContentProvider4	1	0	0	1	1	0	0	1	0	0
	32	Content provider	ntcContentProvider5 + ntcContentProvider6	1	0	0	1	1	0	0	1	0	0
	33	Dynamic Content	ntcDynamicContent1 + ntcDynamicContent2	1	0	0	1	0	0	1	0	0	1
	34	Dynamic Content	ntcDynamicContent3 + ntcDynamicContent4 + ntcDynamicContent5	1	0	0	1	0	0	1	0	0	1
	35	Dynamic Content	ntcDynamicContent6 + ntcDynamicContent7	1	0	0	1	0	0	1	0	0	1
	36	Dynamic Content	ntcDynamicContent8 + ntcDynamicContent9 + ntcDynamicContent10	1	0	0	1	0	0	1	0	0	1
	37	Inter Layer	ntcInterLayer1 + ntcInterLayer2	1	0	0	1	0	1	1	0	1	1
	38	Inter Layer	ntcInterLayer3 + ntcInterLayer4	1	0	0	1	0	1	1	0	1	1
Tổng		Tổng số		55	16	5	37	47	7	6	49	5	6
		Precision p = TP/(TP+FP) = ☑/(☑+☒)			76,19%			87,04%			90,74%		
		Recall r = TP/(TP+FN) = ☑/(☑+☐)			30,19%			88,68%			89,09%		
		F1 measure = 2pr/(p+r)			43,24%			87,85%			89,91%		

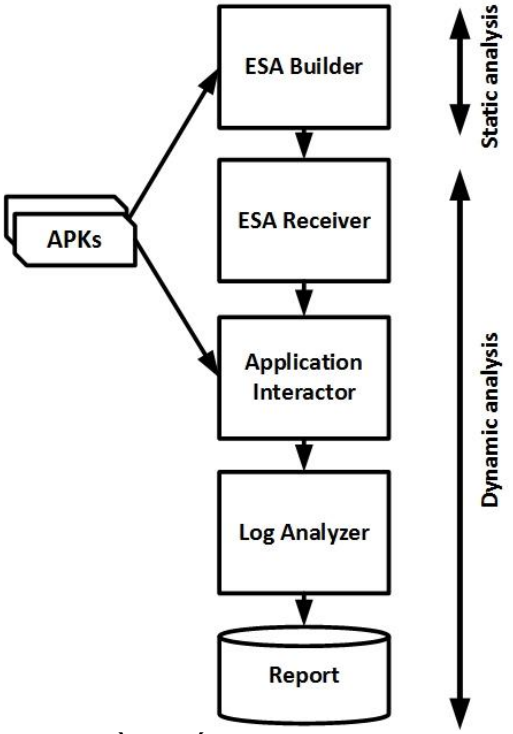
Do DidFail chỉ phân tích thành phần ứng dụng Activity, do đó nghiên cứu này thất bại trong việc phân tích 24 tình huống sử dụng Service, Broadcast Receiver, Content Provider và Shared file trong bộ dữ liệu thử nghiệm uitIACDroid-Bench. Bên cạnh đó, DidFail chỉ phân tích các Intent thuộc dạng Implicit Intent do đó thất bại trong việc phân tích hai tình huống Activity sử dụng Explicit Intent.

Trong số 6 tình huống thử nghiệm của nhóm ứng dụng sử dụng Activity trong bộ dữ liệu thử nghiệm uitIACDroid-Bench có 2 tình huống (tình huống 7, 8 trong Bảng 5) các luồng dữ liệu bị gián đoạn do không thỏa mô hình quyền hạn. Do đó DidFail và IACDroid thất bại trong việc phát hiện hai tình huống này vì các nghiên cứu này chưa xét đến yếu tố quyền hạn. Trong khi đó, hệ thống eddLeak tỏ ra hiệu quả trong việc phát hiện hai tình huống này. Vì hướng tiếp cận này hướng đến tăng độ chính xác, giảm tỷ lệ False Positive nên kết quả thử nghiệm cho thấy eddLeak đáp ứng được mục tiêu. Tuy nhiên, do phải sử dụng thêm yếu tố quyền hạn trong quá trình phân tích, nên eddLeak tốn nhiều thời gian hơn trong quá trình phân tích. Kết quả thực nghiệm cho thấy thời gian trung bình eddLeak tốn nhiều hơn 9.2% so với IACDroid khi chạy trên cùng hệ thống thử nghiệm.

Các hệ thống đều thất bại trong 4 tình huống Dynamic Content, loại tình huống mà các mẫu thử sử dụng cơ chế phát sinh thông tin giao tiếp liên ứng dụng động lúc chạy chương trình. Trong khi đó, 25 tình huống còn lại, hệ thống IACDroid và eddLeak tỏ ra vượt trội so với công trình liên quan vì phân tích cả các thành phần Service, Broadcast Receiver, Content Provider và tập tin chia sẻ. Ngoài ra, các đề xuất này của luận án dùng kỹ thuật phân tích *Incremental analysis* với cơ sở lý thuyết dựa trên đồ thị hai phía có hướng do đó nó có thể phát hiện các luồng dữ liệu qua chuỗi nhiều ứng dụng. Trong luận án này, chúng tôi thử nghiệm thành công giải pháp đề xuất cho tình huống gây rò rỉ thông tin nhạy cảm qua chuỗi 200 ứng dụng với mỗi ứng dụng có 10 luồng dữ liệu. Đây cũng là điểm mạnh của phương pháp đề xuất so với nghiên cứu liên quan hiện tại [63] (hệ thống cho phép phân tích luồng dữ liệu qua chuỗi hai (N=2) ứng dụng).

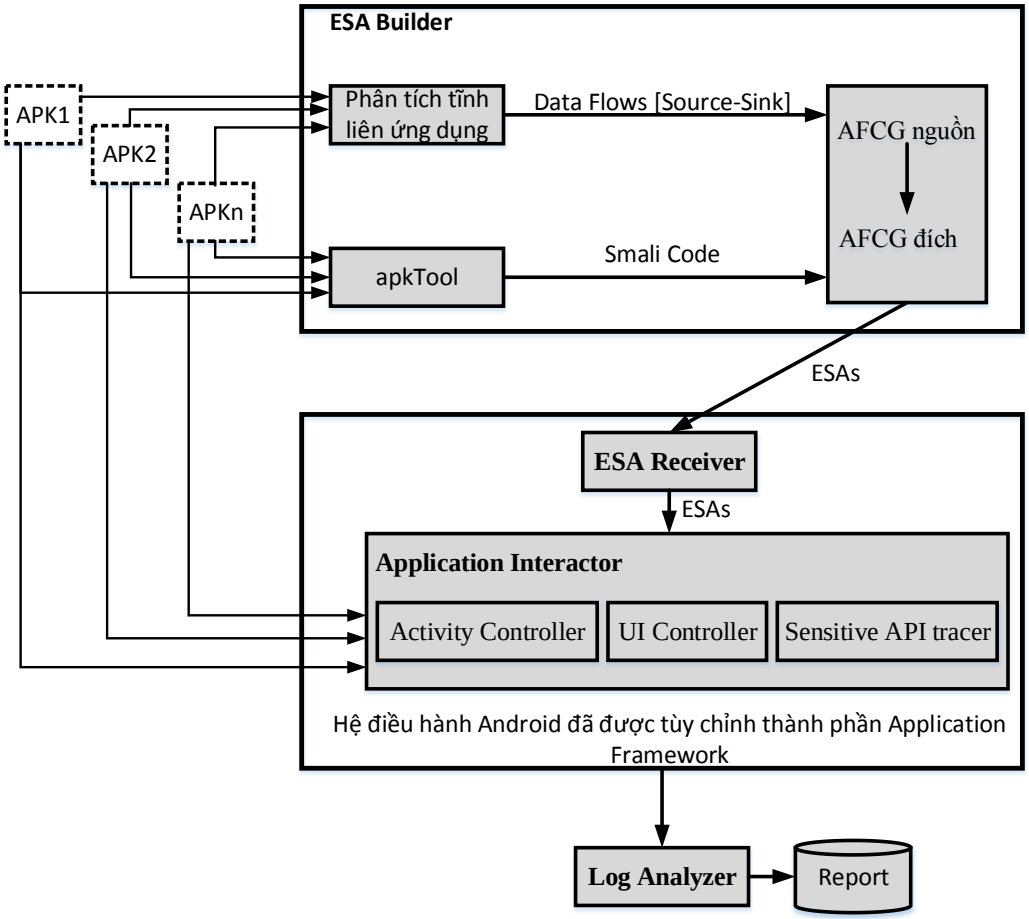
3.3.1.3 Đề xuất phương pháp cải thiện độ chính xác của kết quả phân tích tĩnh bằng kỹ thuật phân tích động

Trong quá trình thực hiện các đề xuất cải tiến độ chính xác của việc phát hiện thất thoát thông tin nhạy cảm qua liên ứng dụng, chúng tôi thấy rằng có một số luồng dữ liệu trong kết quả phân tích không thật sự tồn tại trong thực tế. Việc kiểm chứng kết quả phân tích tĩnh là cần thiết để xác định chính xác các luồng dữ liệu nhạy cảm thật sự tồn tại. Trong phần này, chúng tôi đề xuất phương pháp (đặc tên là eDSDroid) sử dụng phân tích động để kiểm tra sự tồn tại của các luồng dữ liệu nhạy cảm từ kết quả của quá trình phân tích tĩnh. Trong đó, quá trình phân tích động được thực hiện từ kết quả của quá trình phân tích tĩnh. Để tương tác với các ứng dụng trong quá trình phân tích động, chúng tôi chỉnh sửa thành phần Applicaition Framework của hệ điều hành Android theo ý tưởng của SmartDroid [32]. Tuy nhiên phương pháp đề xuất được cải tiến để phân tích được các luồng dữ liệu liên ứng dụng thay vì đơn ứng dụng như SmartDroid. Sơ đồ hệ thống của phương pháp eDSDroid được trình bày trong Hình 15. Nội dung của phương pháp eDSDroid được công bố trong Tạp chí ClusterComputing (Springer, ISI, IF=2.04) [CB2]



Hình 15. Sơ đồ hệ thống của phương pháp eDSDroid

Hệ thống đề xuất có 4 mô-đun chính *ESABuilder*, *ESAReceiver*, *Application Interactor* và *Log Analyzer* được thực hiện qua giai đoạn phân tích tĩnh (static analysis) và phân tích động (dynamic analysis). Trong đó giai đoạn phân tích động được phát triển từ ý tưởng của SmartDroid [32].



Hình 16. Chi tiết sơ đồ hệ thống của eDSDroid

Mô-đun ***ESA Builder*** (Expected Sensitive Action Builder) thực hiện quá trình phân tích tĩnh để rút trích các chuỗi hành động liên quan đến việc gây ra các luồng dữ liệu nhạy cảm trên liên ứng dụng. Trong nghiên cứu này chúng tôi gọi là ESAs (Expected Sensitive Actions).

Mô-đun ***ESA Receiver*** (Expected Sensitive Action Receiver) là một ứng dụng dùng để nhận *ESA* từ *ESA Builder* trước khi gửi đến mô-đun ***Application Interactor***.

Mô-đun ***Application Interactor*** được dùng để tương tác với các ứng dụng thông qua các chuỗi hành động trong ESAs. Bên cạnh đó, chúng tôi thu thập thông tin liên quan đến các hàm nhạy cảm trong quá trình ương tác với các ứng dụng được kiểm tra. Trong nghiên cứu này, chúng tôi chỉnh sửa thành phần Application Framework của hệ điều hành Android để thực hiện quá trình tương tác với các ứng dụng và thu thập thông tin hàm nhạy cảm thay vì sử dụng Xposed Framework [64] và AndroidViewClient [65]. Mô-đun này có ba thành phần con, bao gồm *Activity Controller*, *UI Controller* và *Sensitive API tracer*. Trong đó, chúng tôi xây dựng *Activity Controller* bằng cách chỉnh sửa lớp *Activity class* của thành phần Application Framework để hạn chế việc tương tác với các Activity không có trong danh sách các Activity trong ESA. *UI Controller* được xây dựng bằng cách chỉnh sửa lớp *View class* trong Application Framework để điều khiển các thành phần giao diện liên quan đến các sự kiện trong ESA. Chúng tôi chỉnh sửa mã nguồn của Application Framework để thu thập thông tin về các hàm nhạy cảm trước khi ghi log các thông tin này. Chức năng này gọi là *Sensitive API Tracer*. Thông tin này được thu thập bởi công cụ Android Logcat.

Mô-đun ***Log Analyzer*** được dùng để thu thập và phân tích các thông tin từ mô-đun *Application Interactor* để xác định các luồng thông tin nhạy cảm có thật sự diễn ra không. Hình 17 trình bày một ví dụ về kết quả phân tích của mô-đun này.

```
Data Flow 9:
    Source: echoer - String.getString()
    Sink: echoer - Log.i()

Application Interactor log:
    ##################eDSDroid Log: Start Flow 9

Data Flow 10:
    Source: SendSMS - LocationManager.getLastKnownLocation()
    Sink: echoer - Log.i()

Application Interactor log:
    ##################eDSDroid Log: Start Flow 10
    Src: org.cert.WriteFile.SubActivity->LocationManager getLastKnownLocation() called
    Sink: org.cert.echoer.MainActivity->Log i(String, String) called with params p1 =
```

Hình 17. Ví dụ về thông tin của kết quả phân tích luồng dữ liệu nhạy cảm

Như trình bày ở trên, hệ thống đề xuất xác định danh sách các luồng thông tin nhạy cảm và chuỗi các sự kiện ESAs tương ứng với các luồng thông tin này bằng kỹ thuật phân tích tĩnh. Tiếp đó, hệ thống tương tác với các ứng dụng liên quan trong các ESAs. Trong quá trình tương tác với các ứng dụng, thông tin của các hàm nhạy cảm trong các luồng thông tin sẽ được ghi lại nếu chúng được triệu gọi. Nếu các hàm này xuất hiện trong dữ liệu log sau khi tương tác với các ứng dụng thì chứng tỏ luồng thông tin nhạy cảm này thật sự diễn ra. Ví dụ trong Hình 17 cho thấy hàm nhạy cảm trong luồng thông tin thứ 10 được triệu gọi trong lúc tương tác với ứng dụng, chứng tỏ luồng này thật sự gây thất thoát thông tin nhạy cảm. Trong khi đó, luồng thông tin thứ 9 không thật sự được gọi trong quá trình phân tích động.

Kết quả thử nghiệm cho thấy, phương pháp eDSDroid giải quyết được hạn chế của ba phương pháp DidFail, IACDroid và eddLeak. Cụ thể, nó không phát hiện nhầm 5 tình huống mà cả DidFail, IACDroid và eddLeak gặp phải. Bảng 6 trình bày kết quả so sánh giữa DidFail, IACDroid, eddLeak và phương pháp eDSDroid được đề xuất. Kết quả thử nghiệm cho thấy, mặc dù độ đo Precision (p) của eDSdroid (96,07%) cao hơn so với các phương pháp đề xuất trước tuy nhiên vẫn còn tình trạng phát hiện nhầm và không phát hiện được các luồng dữ liệu liên lớp thành phần. Do eDSDroid thự hiện quá trình phân tích tĩnh trước nên không giải quyết được trường hợp các thông tin hàm giao tiếp liên ứng dụng phát sinh động. Dẫn đến thất bại

trong nhóm tình huống Dynamic Content. Điều này, là động lực để chúng tôi thực hiện đề xuất kế tiếp trong phần sau.

Bảng 6. Kết quả so sánh giữa phương pháp eDSDroid với DidFail, IACDroid và eddLeak

					DidFail			IACDroid			eddLeak			eDSDroid		
Data Set	**STT**	**Nhóm tình huống**	**Tình huống**	**Số luồng thực tế**	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative
DroidBench	1	Activity	SendSMS + StartActivityForResult1 + Echoer	10	10	1	0	10	1	0	10	1	0	10	0	0
uitIACDroid-Bench	2	Activity	ntcIntentActivity1 + ntcIntentActivity2 + ntcIntentActivity3 + ntcIntentActivity4	3	0	0	3	3	0	0	3	0	0	3	0	0
	3	Activity	ntcIntentActivity5 + ntcIntentActivity6 + ntcIntentActivity7 + ntcIntentActivity8	3	0	0	3	3	0	0	3	0	0	3	0	0
	4	Activity	ntcIntentActivity9 + ntcIntentActivity10 + ntcIntentActivity11 + ntcIntentActivity12	4	4	1	0	4	1	0	4	1	0	4	0	0
	5	Activity	ntcIntentActivity13 + ntcIntentActivity14 + ntcIntentActivity15	2	2	1	0	2	1	0	2	1	0	2	0	0
	6	Activity	ntcIntentActivityPermission1 + ntcIntentActivityPermission2 + ntcIntentActivityPermission3 + ntcIntentActivityPermission4	1	0	1	0	0	1	0	1	0	0	1	0	0
	7	Activity	ntcIntentActivityPermission5 + ntcIntentActivityPermission6 + ntcIntentActivityPe	1	0	1	0	0	1	0	1	0	0	1	0	0

			rmission7 + ntcIntentActivityPermission8													
	8	Service	ntcService1 + ntcService2 + ntcService3	1	0	0	1	1	0	0	1	0	0	1	0	0
	9	Service	ntcService4 + ntcService5 + ntcService6	1	0	0	1	1	0	0	1	0	0	1	0	0
	10	Service	ntcService7 + ntcService8 + ntcService9 + ntcService10	1	0	0	1	1	0	0	1	0	0	1	0	0
	11	Service	ntcService11 + ntcService12 + ntcService13 + ntcService14	1	0	0	1	1	0	0	1	0	0	1	0	0
	12	Service	ntcService15 + ntcService16	1	0	0	1	1	0	0	1	0	0	1	0	0
	13	Service	ntcService17 + ntcService18	1	0	0	1	1	0	0	1	0	0	1	0	0
	14	Broadcast Receiver	ntcBroadcast1 + ntcBroadcast2 + ntcBroadcast3	1	0	0	1	1	0	0	1	0	0	1	0	0
	15	Broadcast Receiver	ntcBroadcast4 + ntcBroadcast5 + ntcBroadcast6	1	0	0	1	1	0	0	1	0	0	1	0	0
	16	Broadcast Receiver	ntcBroadcast7 + ntcBroadcast8 + ntcBroadcast9	1	0	0	1	1	0	0	1	0	0	1	0	0
	17	Broadcast Receiver	ntcBroadcast10 + ntcBroadcast11 + ntcBroadcast12	1	0	0	1	1	0	0	1	0	0	1	0	0
	18	Broadcast Receiver	ntcBroadcast13 + ntcBroadcast14 + ntcBroadcast15	1	0	0	1	1	0	0	1	0	0	1	0	0
	19	Broadcast Receiver	ntcBroadcast16 + ntcBroadcast17 + ntcBroadcast18	1	0	0	1	1	0	0	1	0	0	1	0	0
	10	Shared file	ntcWriteFile1 + ntcReadFile2 + ntcReadFile1	1	0	0	1	1	0	0	1	0	0	1	0	0
	21	Shared file	ntcWriteFile3 + ntcReadFile3 + ntcWriteFile4	1	0	0	1	1	0	0	1	0	0	1	0	0
	22	Shared file	ntcWriteFile5 + ntcReadFile5	1	0	0	1	1	0	0	1	0	0	1	0	0
	23	Shared file	ntcWriteFile6 + ntcReadFile6	1	0	0	1	1	0	0	1	0	0	1	0	0
	24	Shared file	ntcWriteFile7 + ntcReadFile7	1	0	0	1	1	0	0	1	0	0	1	0	0
	25	Shared file	ntcWriteFile8 + ntcReadFile8 + ntcReadAndWriteFile8	1	0	0	1	1	0	0	1	0	0	1	0	0
	26	Shared file	ntcWriteFile9 + ntcReadFile9 + ntcReadAndWriteFile9	1	0	0	1	1	0	0	1	0	0	1	0	0
	27	Shared file	ntcWriteFile10 +	1	0	0	1	1	0	0	1	0	0	1	0	0

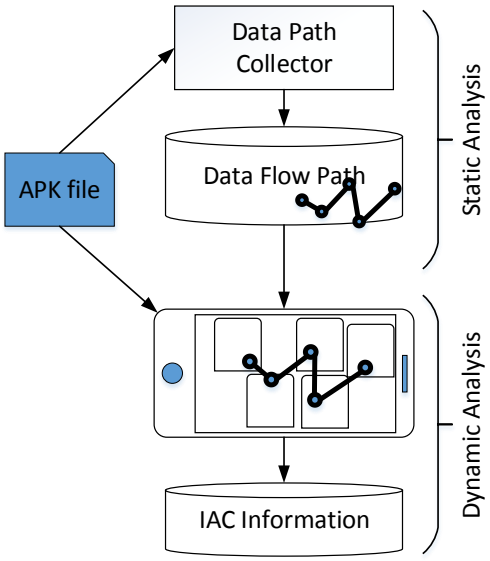
			ntcReadFile10 + ntcReadAndWriteFile10													
	28	Shared file	ntcWriteFile11 + ntcReadFile11 + ntcReadAndWriteFile11	1	0	0	1	1	0	0	1	0	0	1	0	0
	29	Shared file	ntcSharedFileChain1+ … + ntcSharedFileChain200	1	0	0	1	1	0	0	1	0	0	1	0	0
	30	Content provider	ntcContentProvider1 + ntcContentProvider2	1	0	0	1	1	0	0	1	0	0	1	0	0
	31	Content provider	ntcContentProvider3 + ntcContentProvider4	1	0	0	1	1	0	0	1	0	0	1	0	0
	32	Content provider	ntcContentProvider5 + ntcContentProvider6	1	0	0	1	1	0	0	1	0	0	1	0	0
	33	Dynamic Content	ntcDynamicContent1 + ntcDynamicContent2	1	0	0	1	0	0	1	0	0	1	0	0	1
	34	Dynamic Content	ntcDynamicContent3 + ntcDynamicContent4 + ntcDynamicContent5	1	0	0	1	0	0	1	0	0	1	0	0	1
	35	Dynamic Content	ntcDynamicContent6 + ntcDynamicContent7	1	0	0	1	0	0	1	0	0	1	0	0	1
	36	Dynamic Content	ntcDynamicContent8 + ntcDynamicContent9 + ntcDynamicContent10	1	0	0	1	0	0	1	0	0	1	0	0	1
	37	Inter Layer	ntcInterLayer1 + ntcInterLayer2	1	0	0	1	0	1	1	0	1	1	0	1	1
	38	Inter Layer	ntcInterLayer3 + ntcInterLayer4	1	0	0	1	0	1	1	0	1	1	0	1	1
Tổng		Tổng số		55	16	5	37	47	7	6	49	5	6	49	2	6
		Precision p = TP/(TP+FP) = ☑/(☑+☒)			76,19%			87,04%			90,74%			96,07%		
		Recall r = TP/(TP+FN) = ☑/(☑+☐)			30,19%			88,68%			89,09%			89,09%		
		F1 measure = 2pr/(p+r)			43,24%			87,85%			89,91%			92,45%		

3.3.1.4 Đề xuất phương pháp cải thiện độ chính xác của việc phân tích luồng dữ liệu liên ứng dụng bằng cách kết hợp phân tích tĩnh và phân tích động

Trong phần trước, chúng tôi đề xuất phương pháp cải thiện độ chính xác của kết quả phân tích tĩnh bằng kỹ thuật phân tích động, tuy nhiên nó vẫn còn gặp một số hạn chế trong việc phân tích các ứng dụng sử dụng kỹ thuật phát sinh dữ liệu động trong quá trình thực hiện luồng dữ liệu nhạy cảm liên ứng dụng. Trong phần này, chúng tôi đề xuất phương pháp (đặt tên là uitHydroid) kết hợp phân tích tĩnh và động để giải quyết các hạn chế này nhằm cải thiện độ chính xác. Hướng tiếp cận này không cần chỉnh sửa mã nguồn của hệ điều hành Android như các công trình [34] và eddLeak. Việc tương tác lên tất cả các thành phần giao diện trong quá trình phân tích động [66] tốn nhiều thời gian. Việc xác định các thành phần giao diện cần tương tác trong quá trình phân tích động như [32] giúp ích giảm thời gian phân tích. Tuy nhiên công trình [32] chỉ phân tích trên ứng dụng đơn. Công việc kế tiếp của luận án là xuất mở rộng cho phép phân tích trên liên ứng dụng. Nội dung của phương pháp này được công bố trong Tạp chí ClusterComputing (Springer, ISI, IF=2.04) [CB1].

Các ứng dụng Android chứa nhiều Activity. Mỗi Activity có chứa nhiều thành phần giao diện. Có nhiều loại thành phần giao diện như Button, Edit Text, Text View,… Người dùng có thể tương tác với các ứng dụng thông qua các thành phần này. Để phát hiện một ứng dụng có gây thất thoát thông tin nhạy cảm hay không, chúng ta phải thực hiện việc tương tác lên ứng dụng để kích hoạt hành vi này của chúng. Việc tương tác này có thể theo cách vét cạn, tức là tương tác lên tất cả các thành phần giao diện. Tuy nhiên việc này tốn nhiều thời gian. Thay vào đó, ta chỉ cần tương tác lên các thành phần giao diện có khả năng gây ra thất thoát thông tin nhạy cảm để rút ngắn thời gian phân tích. Ý tưởng chính của phương pháp này là dùng phân tích tĩnh để xác định thành phần giao diện liên quan đến các luồng dữ liệu. Thông tin có được từ quá trình phân tích tĩnh sẽ giúp định hướng quá trình phân tích động để quá trình phân tích động diễn ra nhanh hơn. Tuy nhiên, trong quá trình phân tích tĩnh, một số đường đi bị gián đoạn do thông tin của các thành phần kế tiếp chỉ có thể xác định dựa vào thông tin người dùng tương tác. Trong phân tích động, uitHyDroid tương tác với các thành phần giao diện liên quan được xác định trong quá trình phân tích tĩnh. Việc tương tác này giúp phát sinh thông tin của thành phần/ứng dụng kế tiếp thông qua hàm thực hiện giao tiếp liên thành phần (ICC) và giao tiếp liên ứng dụng (IAC). Việc thu thập các thông tin ICC và IAC trong lúc phân tích động được thực hiện bởi một mô-đun sử dụng kỹ thuật hook (Xposed Framework [67]). Điều này cho phép phân tích động mà không cần phải chỉnh sửa mã nguồn của hệ điều hành Android.

Hình 18 trình bày sơ đồ hệ thống của phương pháp uitHyDroid. Hệ thống có ba phần chính: **Data Path Collector**, **Automation Operator** và **IAC Collector**. Data Path Collector là thành phần xác định các thành phần giao diện cần tương tác theo các luồng dữ liệu được xác định trong quá trình phân tích tĩnh. Trong khi đó, Automation Operator tiến hành việc tương tác lên các thành phần giao diện, còn IAC Collector thực hiện việc thu thập thông tin của các giao tiếp liên ứng dụng để hỗ trợ trong việc xác định các thành phần ứng dụng hoặc ứng dụng kế tiếp trong quá trình phân tích động.



Hình 18. Sơ đồ hệ thống của phương pháp uitHydroid

Kết quả thử nghiệm phương pháp đề xuất được trình bày trong Bảng 7 cho thấy phương pháp uitHyDroid chính xác hơn eDSDroid trong việc phân tích các tình huống phát sinh thông tin giao tiếp liên ứng dụng động. Kết quả thử nghiệm cho thấy, mặc dù độ đo Precision (*p*) của uitHydroid (96,36%) cao hơn so với các phương pháp đề xuất trước tuy nhiên vẫn còn tình trạng phát hiện nhầm và không phát hiện được các luồng dữ liệu liên lớp thành phần. Đây là hướng nghiên cứu trong tương lai của luận án.

Bảng 7. Kết quả thử nghiệm phương pháp uitHyDroid so sánh với DidFail, IACDroid, eddLeak và eDSDroid

					DidFail			IACDroid			eddLeak			eDSDroid			uitHydroid		
Data Set	**STT**	**Nhóm tình huống**	**Tình huống**	**Số luồng thực tế**	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative	☑ True Positive	☒ False Positive	☐ False Negative
DroidBench	1	Activity	SendSMS + StartActivity ForResult1 + Echoer	10	10	1	0	10	1	0	10	1	0	10	0	0	10	0	0
uitIACDroid-Bench	2	Activity	ntcIntentActivity1 + ntcIntentActivity2 + ntcIntentActivity3 + ntcIntentActivity4	3	0	0	3	3	0	0	3	0	0	3	0	0	3	0	0
	3	Activity	ntcIntentActivity5 + ntcIntentActivity6 + ntcIntentActivity7 + ntcIntentActivity8	3	0	0	3	3	0	0	3	0	0	3	0	0	3	0	0

4	Activity	ntcIntentActivity9 + ntcIntentActivity10 + ntcIntentActivity11 + ntcIntentActivity12	4	4	1	0	4	1	0	4	1	0	4	0	0	4	0	0
5	Activity	ntcIntentActivity13 + ntcIntentActivity14 + ntcIntentActivity15	2	2	1	0	2	1	0	2	1	0	2	0	0	2	0	0
6	Activity	ntcIntentActivityPermission1 + ntcIntentActivityPermission2 + ntcIntentActivityPermission3 + ntcIntentActivityPermission4	1	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0
7	Activity	ntcIntentActivityPermission5 + ntcIntentActivityPermission6 + ntcIntentActivityPermission7 + ntcIntentActivityPermission8	1	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0
8	Service	ntcService1 + ntcService2 + ntcService3	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
9	Service	ntcService4 + ntcService5 + ntcService6	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
10	Service	ntcService7 + ntcService8 + ntcService9 + ntcService10	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
11	Service	ntcService11 + ntcService12 + ntcService13 + ntcService14	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
12	Service	ntcService15 + ntcService16	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
13	Service	ntcService17 +	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0

			ntcService18																
	14	Broadcast Receiver	ntcBroadcast 1 + ntcBroadcast 2 + ntcBroadcast 3	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	15	Broadcast Receiver	ntcBroadcast 4 + ntcBroadcast 5 + ntcBroadcast 6	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	16	Broadcast Receiver	ntcBroadcast 7 + ntcBroadcast 8 + ntcBroadcast 9	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	17	Broadcast Receiver	ntcBroadcast 10 + ntcBroadcast 11 + ntcBroadcast 12	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	18	Broadcast Receiver	ntcBroadcast 13 + ntcBroadcast 14 + ntcBroadcast 15	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	19	Broadcast Receiver	ntcBroadcast 16 + ntcBroadcast 17 + ntcBroadcast 18	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	10	Shared file	ntcWriteFile1 + ntcReadFile2 + ntcReadFile1	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	21	Shared file	ntcWriteFile3 + ntcReadFile3 + ntcWriteFile4	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	22	Shared file	ntcWriteFile5 + ntcReadFile5	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	23	Shared file	ntcWriteFile6 + ntcReadFile6	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	24	Shared file	ntcWriteFile7 + ntcReadFile7	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	25	Shared file	ntcWriteFile8 + ntcReadFile8 +	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0

			ntcReadAndWriteFile8																
	26	Shared file	ntcWriteFile9 + ntcReadFile9 + ntcReadAndWriteFile9	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	27	Shared file	ntcWriteFile10 + ntcReadFile10 + ntcReadAndWriteFile10	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	28	Shared file	ntcWriteFile11 + ntcReadFile11 + ntcReadAndWriteFile11	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	29	Shared file	ntcSharedFileChain1+ … + ntcSharedFileChain200	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	30	Content provider	ntcContentProvider1 + ntcContentProvider2	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	31	Content provider	ntcContentProvider3 + ntcContentProvider4	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	32	Content provider	ntcContentProvider5 + ntcContentProvider6	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
	33	Dynamic Content	ntcDynamicContent1 + ntcDynamicContent2	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0
	34	Dynamic Content	ntcDynamicContent3 + ntcDynamicContent4 + ntcDynamicContent5	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0
	35	Dynamic Content	ntcDynamicContent6 + ntcDynamicContent7	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0
	36	Dynamic Content	ntcDynamicContent8 + ntcDynamicContent9 + ntcDynamicContent10	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0
	37	Inter Layer	ntcInterLayer1 + ntcInterLayer2	1	0	0	1	0	1	1	0	1	1	0	1	1	0	1	1

	38	Inter Layer	ntcInterLayer 3 + ntcInterLayer 4	1	0	0	1	0	1	1	0	1	1	0	1	1	0	1	1
Tổng		Tổng số		55	16	5	37	47	7	6	49	5	6	49	2	6	53	2	2
		Precision p = TP/(TP+FP) = ☑/(☑+☒)			76,19%			87,04%			90,74%			96,07%			96,36%		
		Recall r = TP/(TP+FN) = ☑/(☑+☐)			30,19%			88,68%			89,09%			89,09%			96,36%		
		F1 measure = 2pr/(p+r)			43,24%			87,85%			89,91%			92,45%			96,36%		

Để đánh giá tính hiệu quả của việc tương tác lên các thành phần giao diện cần thiết thay vì tương tác lên tất cả, chúng tôi định nghĩa một số công thức sau:

Gọi t là thời gian thao tác trên một thành phần giao diện.

Gọi N_b là số thành phần giao diện của toàn ứng dụng.

Gọi T_b là thời gian để thao tác trên tất cả các thành phần giao diện.

$$T_b = t \times N_b \quad (1)$$

Gọi Na là số thành phần giao diện mà các đường dữ liệu đi qua.

$$N_a \leq N_b \quad (2)$$

Gọi T_a là thời gian để thao tác trên các thành phần giao diện theo các đường dữ liệu đi qua.

$$T_a = t \times N_a \quad (3)$$

Từ (1), (2) và (3) ta có: $T_a \leq T_b \quad (4)$

Từ công thức (4) ta thấy việc kiểm thử các trường hợp rò rỉ thông tin bằng cách thao tác trên các thành phần giao diện liên quan đến các đường đi của dữ liệu nhạy cảm sẽ nhanh hơn so với việc thao tác trên tất cả các thành phần giao diện như một số giải pháp trước đây [68].

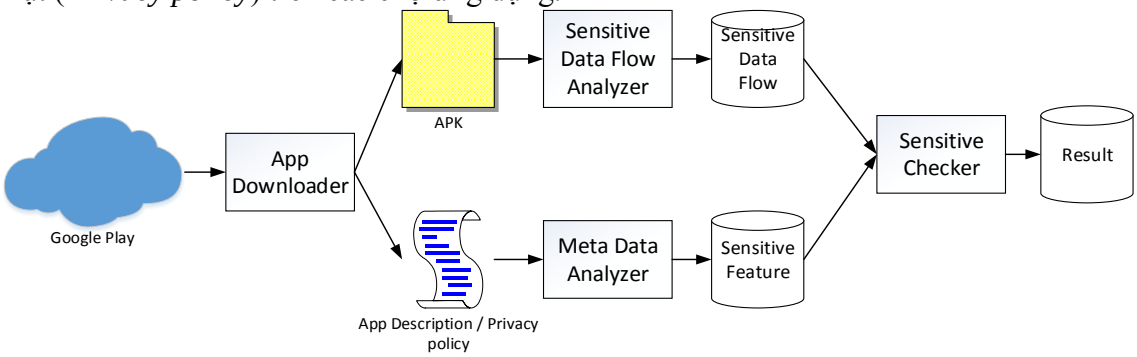
Trong nghiên cứu này, chúng tôi gọi tỷ số $f = \frac{N_a}{N_b}$ biểu diễn tính hiệu quả của việc tương tác theo kịch bản thay vì tương tác theo thuật toán duyệt theo chiều sâu (vét cạn). Tỷ số f càng nhỏ chứng tỏ hiệu quả của việc tương tác theo kịch bản càng cao. Kết quả phân tích dữ liệu thử nghiệm cho thấy tỷ lệ f có giá trị thấp. Giá trị trung bình của f là $\bar{f} = 37.22\%$. Kết quả này cho thấy, tỷ lệ các thành phần giao diện cần được tương tác trên mỗi ứng dụng là khá ít so với tổng số các thành phần giao diện trong trường hợp muốn kích hoạt các hành động gây rò rỉ thông tin nhạy cảm. Phát hiện này cho thấy sự cần thiết phải thực hiện bước phân tích tĩnh để định hướng cho việc phân tích động hiệu quả hơn.

Để đánh giá hiệu quả về mặt thời gian trong quá trình thực hiện phân tích, chúng tôi thử nghiệm phương pháp đề xuất và hướng tiếp cận tương tác lên tất cả các thành phần giao diện theo thuật toán duyệt theo chiều sâu. Thời gian phân tích của hướng tiếp cận đề xuất bao gồm thời gian của quá trình phân tích tĩnh và phân tích động, trong khi đó thời gian phân tích của hướng tiếp cận duyệt theo chiều sâu chỉ bao gồm thời gian thực hiện quá trình phân tích động. Tuy nhiên, kết quả thử nghiệm cho thấy thời gian phân tích trung bình theo hướng tiếp cận của luận án bằng 42% so với hướng tiếp cận duyệt theo chiều sâu. Như vậy giải pháp đề xuất hiệu quả hơn so với các nghiên cứu liên quan hiện tại, cả về khả năng phân tích liên ứng dụng và tốc độ.

Mặc dù vậy, trong các đề xuất trước của luận án cũng như các công trình liên quan, chúng tập trung vào việc phát hiện thất thoát thông tin trong một hoặc một nhóm ứng dụng. Chúng chưa xét đến ngữ cảnh sử dụng ứng dụng đó, tức chưa xét đến chức năng của các ứng dụng khi phân tích. Tuy nhiên, trong thực tế, chúng ta thấy rằng có một số ứng dụng được phát triển để thực hiện một số chức năng cụ thể, trong số đó có thể có các chức năng liên quan đến việc gửi thông tin nhạy cảm ra khỏi thiết bị. Ví dụ, ứng dụng gọi xe taxi sẽ thu thập vị trí của hành khách và gửi đến máy chủ trước khi chia sẻ cho tài xế. Trong trường hợp này, ứng dụng gọi xe sẽ được đánh dấu là gây thất thoát thông tin bởi các kỹ thuật phát hiện hiện tại mặc dù ứng dụng này được viết ra là để thực hiện điều đó. Như vậy, khi phát hiện một ứng dụng gây thất

thoát thông tin chúng ta cần thiết phải kiểm tra xem hành vi đó có phải là chức năng được mô tả của ứng dụng hay không trước khi cảnh báo cho người dùng.

Nhận thức được vấn đề này trong quá trình thực hiện luận án, chúng tôi bước đầu đề xuất phương pháp giảm tỷ lệ phát hiện nhầm đối với các ứng dụng có chức năng gửi thông tin ra khỏi thiết bị bằng cách phân tích nội dung của thông tin mô tả của ứng dụng và hành vi khi phân tích tập tin APK của chúng như Hình 19. Với giả định rằng, các nhà phát triển ứng dụng sẽ mô tả các chức năng của ứng dụng trong phần mô tả (*App Description*) và chính sách bảo mật (*Privacy policy*) trên các chợ ứng dụng.



Hình 19. Hệ thống kiểm tra mối liên hệ giữa thông tin mô tả và hành vi của ứng dụng

Trong hệ thống đề xuất này, mô-đun *App Downloader* chịu trách nhiệm tải các dữ liệu dạng meta data của ứng dụng trên chợ ứng dụng như thông tin mô tả, chính sách bảo mật và tập tin cài đặt (apk) của ứng dụng. Mô-đun *Meta Data Analyzer* chịu trách nhiệm phân tích thông tin mô tả và chính sách bảo mật của ứng dụng để rút trích các hành vi chức năng (*Sensitive Feature*) của ứng dụng (ví dụ: danh sách hàm truy cập dữ liệu nhạy cảm, hàm gửi dữ liệu ra khỏi thiết bị và quyền hạn tương ứng với các hàm này). Thông tin về *Sensitve Feature* từ mô-đun *Meta Data Analyzer* được dùng để kiểm chứng các kết quả phân tích của mô-đun phân tích tập tin cài đặt (APK) của ứng dụng. Mô-đun *Sensitive Checker* sẽ chịu trách nhiệm kiểm tra kết quả phân tích của mô-đun *Sensitive Data Flow* là *Sensitive Data Flow* có nằm trong nhóm hành vi (*Sensitive Feature*) được mô tả trong thông tin mô tả và chính sách bảo mật hay không. Nếu có, tức là việc thất thoát thông tin của ứng dụng là chức năng bình thường của ứng dụng lúc đó hệ thống chỉ ghi nhận mà không cảnh báo cho người dùng. Ngược lại, việc thất thoát thông tin của ứng dụng là không bình thường và cần phải cảnh báo cho người dùng.

Một phần nội dung của đề xuất này đã được công bố trong tạp chí Thông tin và truyền thông (CB9). Vì mảng xử lý ngôn ngữ tự nhiên không phải là thế mạnh của chúng tôi, nên đề xuất này của chúng tôi chỉ dừng lại ở mức đề xuất mô hình và sử dụng các kỹ thuật xử lý ngôn ngữ tự nhiên đơn giản. Do đó kết quả của việc phân tích mô tả ứng dụng bằng xử lý ngôn ngữ tự nhiên còn hạn chế. Tuy nhiên, hướng tiếp cận này hứa hẹn nhiều kết quả khả quan trong tương lai nếu được kết hợp với các nhóm nghiên cứu mạnh về xử lý ngôn ngữ tự nhiên.

3.3.2 Đề xuất phương pháp phát hiện thất thoát thông tin nhạy cảm trong thành phần Application Framework của hệ điều hành Android

Với giả định rằng, thành phần ứng dụng cài sẵn an toàn, hoặc thậm chí loại bỏ các ứng dụng cài sẵn trong các firmware Android thì vẫn chưa thể kết luận nó an toàn, vì nguy cơ gây thất thoát thông tin nhạy cảm có thể tồn tại trong các thành phần khác như Application Framework, Linux Kernel. Chính vì thế việc phân tích thành phần Application Framework có ý nghĩa quan trọng trong bối cảnh này. Trong phần này, chúng tôi đề xuất phương pháp phát hiện thất thoát thông tin nhạy cảm trong thành phần Application Framework của hệ điều hành Android bằng cách áp dụng phương pháp phân tích luồng dữ liệu trên ứng dụng. Một phần nội dung của phương pháp này được trình bày tại hội nghị FAIR2017 [CB5] và MDM2017 [CB6].

Thành phần Application Framework trong hệ điều hành Android chứa nhiều lớp liên quan. Khi khởi động hệ thống, các lớp này được tải lên bộ nhớ để các ứng dụng có thể sử dụng các chức năng cung cấp bởi các lớp này. Tương tự như cách thức phân tích các tập tin APK. Chúng

ta cần phải xác định điểm bắt đầu cho Application Framework (giống như hàm *main* trong ứng dụng Java). Trong nghiên cứu này, để tạo điểm bắt đầu cho quá trình phân tích luồng dữ liệu trong Application Framework, chúng tôi tạo hàm *main* ảo cho các hàm trong Applicaition Framework. Tuy nhiên, không giống như các ứng dụng Android, Application Framework Android có rất nhiều hàm (mỗi Application Framework có trung bình 65,100 hàm, tùy từng phiên bản của Hệ điều hành Android). Do đó, chúng tôi tiến hành lựa chọn các hàm khác biệt giữa Application Framework cần phân tích và Application Framework từ firmware chuẩn của Google trước khi tiến hành tạo hàm *main* ảo. Quy trình được thực hiện theo ba bước chính:

Bước 1: Liệt kê các hàm khác biệt so với các hàm trong firmware Android chuẩn. Trong nghiên cứu này, firmware chuẩn được hiểu là firmware từ Google hoặc là firmware đã được đánh dấu là an toàn trong những lần phân tích trước.

Bước 2: Tạo hàm *main* ảo cho các hàm được lựa chọn ở bước 1.

Bước 3: Sau khi liệt kê được các hàm khác biệt và tạo được hàm *main* ảo, chúng tôi tiến hành phân tích luồng thông tin nhạy cảm trong Application Framework như phương pháp phân tích luồng dữ liệu nhạy cảm bằng kỹ thuật phân tích tĩnh đã trình bày trong đề xuất trên.

Trong nghiên cứu này, chúng tôi lần lượt thử nghiệm phương pháp đề xuất với bộ dữ liệu thử nghiệm uitCustomROM-Bench do nhóm tự phát triển được mô tả trong Bảng 3 và 290 firmware Android tải từ Internet.

Hệ thống đề xuất được triển khai trên hệ thống máy tính với cấu hình phần cứng được trình bày trong Bảng 8.

Bảng 8. Cấu hình phần cứng của hệ thống thử nghiệm

Thành phần	Thông số cấu hình
CPU	Intel Core i7-4720HQ
Memory	16 GB
HDD	1TB, 7200 rpm

Kết quả thử nghiệm cho thấy hệ thống phát hiện chính xác các trường hợp gây thất thoát thông tin nhạy cảm tầng Application Framework trong các firmware Android tùy biến do nhóm tự phát triển.

Listing 1 mô tả một kết quả phân tích của phương pháp đề xuất đối với mẫu thử thứ 5 trong Bảng 3. Kết quả phân tích này cho thấy, hệ thống phát hiện đúng trường hợp thất thoát thông tin nhạy cảm trong thành phần Application Framework.

Listing 1: Một ví dụ của kết quả phân tích tầng Application Framework của ROM Android


```
{
  "analyzeTime": {
    "date": {
      "year": 2017,
      "month": 4,
      "day": 27
    },
    "time": {
      "hour": 14,
      "minute": 33,
      "second": 50,
      "nano": 60000000
    }
  },
  "totalDuration": {
    "seconds": 19,
    "nanos": 671000000
  },
  "systemInfo": {
    "memorySpeed": "2133",
    "os": "Microsoft Windows 10 Pro",
    "availableMemory": "9566780",
```

```
        "processor": "Intel(R) Core(TM) i7-4720HQ CPU @ 2.70GHz"
      },
      "frameworkName": "Leak_IMEI_Socket.jar",
      "frameworkHash":
"1216c927c6d7206d1007fef5bd269f06e059a5740b667eeba129cd25f353fec7",
      "sizeInByte": 6576366,
      "numberOfAnalyzeMethods": 6,
      "analyzeMethods": [
        "<android.hardware.camera2.CameraManager:
android.os.PowerManager$WakeLock -
get0(android.hardware.camera2.CameraManager)>",
        "<android.hardware.camera2.CameraManager$1: void
onTorchModeChanged(java.lang.String,boolean)>",
        "<android.hardware.camera2.CameraManager$1: void
onTorchModeUnavailable(java.lang.String)>",
        "<android.hardware.camera2.CameraManager$1: void
<init>(android.hardware.camera2.CameraManager)>",
        "<android.hardware.camera2.CameraManager: void
<init>(android.content.Context)>",
        "<android.app.Activity: void onCreate(android.os.Bundle)>"
      ],
      "numberOfFlow": 1,
      "flows": [
        {
          "source": "$r9 = virtualinvoke
$r19.<android.telephony.TelephonyManager: java.lang.String
getDeviceId()>()",
          "sink": "virtualinvoke $r2.<java.io.DataOutputStream: void
writeUTF(java.lang.String)>($r9)",
          "sourceMethod": "<android.telephony.TelephonyManager:
java.lang.String getDeviceId()>",
          "sinkMethod": "<java.io.DataOutputStream: void
writeUTF(java.lang.String)>",
          "pathLength": 2,
          "path": [
            "virtualinvoke $r2.<java.io.DataOutputStream: void
writeUTF(java.lang.String)>($r9)",
            "$r9 = virtualinvoke
$r19.<android.telephony.TelephonyManager: java.lang.String
getDeviceId()>()"
          ]
        }
      ]
    }
```


Để thử nghiệm khả năng phân tích các firmware Android tùy biến từ các diễn đàn chia sẻ trên Internet, chúng tôi tiến hành phân tích 290 firmware Android tùy biến được tải từ các nguồn khác nhau. Trong giới hạn về số lượng firmware Android tùy biến được tải này, chúng tôi chưa phát hiện nguy cơ bảo mật ở mức Application Framework trong thế giới thực. Kết quả phân tích được trình bày trong Bảng 9.

Bảng 9. Kết quả rút trích thành phần Linux Kernel

Tên hãng	Số lượng firmware	Số ứng dụng trung bình	Tình huống rò rỉ thông tin trong Application Framework	Thời gian rút trích các thành phần (giây)	Thời gian phân tích Application Framework

uitCustomROM Bench	10	348	Có	78,01	19,62
Asus	27	116	Không	90,21	20,11
LG	32	102	Không	140,33	19,22
Motorola	28	77	Không	70,49	18,92
Nexus	24	160	Không	121,42	20,01
Samsung	30	109	Không	118,87	19,88
Sony	29	85	Không	79,99	21,72
Khác	120	125	Không	88,52	20,92

Như vậy, với các đề xuất này, hướng tiếp cận của luận án có thể cho phép phát hiện thất thoát thông tin nhạy cảm qua nhiều ứng dụng và thất thoát thông tin nhạy cảm trong thành phần Application Framework của hệ điều hành Android. Nghiên cứu tương lai của luận án là mở rộng phân tích các loại giao tiếp covert chanel, phân tích thành phần khác của hệ điều hành Android như Linux Kernel và phân tích mối quan hệ giữa các thành phần thay vì phân tích từng thành phần riêng biệt như hiện tại để phát hiện các luồng dữ liệu nhạy cảm liên lớp thành phần.

Chương 4. KẾT LUẬN

4.1 Các kết quả đạt được

Luận án có ba đóng góp mới chính như sau:

Thứ nhất, luận án đề xuất phương pháp cải thiện độ chính xác của việc phát hiện thất thoát thông tin nhạy cảm trong các ứng dụng Android. Cụ thể phương pháp đề xuất cho phép mở rộng các loại giao tiếp liên ứng dụng, kiểm tra khả năng tồn tại của các giao tiếp liên ứng dụng, tiết kiệm thời gian trong quá trình phân tích động. Một phần nội dung của đóng góp này được đăng trong tạp chí Cluster Computing (Springer, ISI, IF: 2.04) [CB1, CB2], kỷ yếu hội nghị ICISA2016 (Springer) [CB3] và ICCSN2017 (IEEE) [CB5].

Thứ hai, luận án đề xuất phương pháp áp dụng kỹ thuật phân tích luồng dữ liệu trong ứng dụng Android để phát hiện thất thoát thông tin nhạy cảm trong thành phần Application Framework trong firmware Android. Một phần nội dung của phương pháp này được trình bày tại IEEE MDM2017 [CB4], FAIR2017 [CB6]

Thứ ba, đóng góp về mặc kỹ thuật trong việc phát triển các công cụ phục vụ cho quá trình thực hiện các đề xuất của luận án. Mặc dù, đóng góp thứ ba mang tính kỹ thuật là chủ yếu, tuy nhiên công sức mà chúng tôi bỏ ra là đáng ghi nhận trong quá trình thực hiện luận án.

Bên cạnh đó, một đóng góp phụ khác của luận án là xây dựng bộ dữ liệu thử nghiệm có số kịch bản chứa đầy đủ trường hợp gây thất thoát thông tin nhạy cảm hơn. Đây được xem là đóng góp giá trị cho cộng đồng nghiên cứu trong lĩnh vực này. Bộ dữ liệu thử nghiệm này nhận được ý kiến phản hồi tích cực từ các nhóm nghiên cứu liên quan. Một phần nội dung của đóng góp này được trình bày tại hội nghị ICCAI2019 [CB8] (kỷ yếu xuất bản bởi ACM).

4.2 Các công bố trong luận án

4.2.1 Các công bố chính của luận án

4.2.1.1 Tạp chí ISI

Trong quá trình thực hiện luận án, nghiên cứu sinh đã tham gia công bố các kết quả chính của luận án trong tạp chí khoa học chuyên ngành nằm trong danh mục ISI (SCIE) được xuất bản bởi Springer với Impact factor là 2.04. Cụ thể:

[CB1] **Nguyen Tan Cam**, Van-Hau Pham, Tuan Nguyen, 2017, "Detecting sensitive data leakage via inter-applications on Android using a hybrid analysis technique", *Cluster Computing - The journal of networks software tools and applications, ISSN: 1386-7857,* ***Springer, ISI (SCIE), Impact factor: 2.04****, (2017)* *https://doi.org/10.1007/s10586-017-1260-2*

[CB2] Ly Hoang Tuan, **Nguyen Tan Cam**, Van-Hau Pham, 2017, "Enhancing the accuracy of static analysis for detecting sensitive data leakage in Android by using dynamic analysis", *Cluster Computing - The journal of networks software tools and applications, ISSN: 1386-7857,* ***Springer, ISI (SCIE), Impact factor: 2.04****, (2017). https://doi.org/10.1007/s10586-017-1364-8*

4.2.1.2 Kỷ yếu hội nghị khoa học

Trong quá trình thực hiện luận án, nghiên cứu sinh đã tham gia báo cáo các kết quả nghiên cứu chính của luận án tại các hội nghị khoa học quốc tế với kỷ yếu được xuất bản bởi các nhà xuất bản Springer và IEEE, cũng như các hội nghị khoa học uy tín trong nước như hội nghị FAIR. Cụ thể:

[CB3] **Nguyen Tan Cam**, Van-Hau Pham, and Tuan Nguyen, 2016, "Android Security Analysis Based on Inter-application Relationships," *in Proceedings of Information Science and Applications (ICISA) 2016,* ***Springer****. DOI: 10.1007/978-981-10-0557-2_68*

[CB4] **Nguyen Tan Cam**, Van-Hau Pham, Tuan Nguyen, 2017, "Sensitive Data Leakage Detection in Pre-Installed Applications of Custom Android Firmware", *in Proceedings of the 18th IEEE International Conference on Mobile Data Management (MDM2017). **IEEE**, DOI 10.1109/MDM.2017.56. **Rank C (ERA)***

[CB5] Phan The Duy, **Nguyen Tan Cam**, Van-Hau Pham, 2017, "eddLeak: Enhancing precision of detecting inter-app data leakage in Android applications", *in Proceeding of 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN2017). **IEEE**. DOI 10.1109/ICCSN.2017.8230197*

[CB6] **Nguyen Tan Cam**, Van-Hau Pham, Tuan Nguyen, 2017, "Detect security threat in android custom firmware by analyzing applications framework and default settings", *in Proceedings of the 10th National Conference on Fundamental and Applied Information Technology Research (**FAIR'10**), ISBN: 978-604-913-614-6, DOI:10.15625/vap.2017.00011.*

4.2.2 Các công bố phụ liên quan đến luận án

Ngoài các công bố chính, chúng tôi công bố các công bố phụ liên quan đến nội dung của luận án. Cụ thể:

[CB7] **Nguyễn Tấn Cầm**, Phạm Văn Hậu, Nguyễn Anh Tuấn, 2017, "Phát hiện nguy cơ gây thất thoát thông tin trên thiết bị Android bằng kỹ thuật phân tích động", *Tạp chí Thông tin và truyền thông, ISSN 1859-3550, trang 17-23, Số 551(741) 12/2017.*

[CB8] Nguyen Tan Cam, Nghi Hoang Khoa, Le Duc Thinh, Van-Hau Pham, Tuan Nguyen, 2019, *Proposing Automatic Dataset Generation System to Support Android Sensitive Data Leakage Detection Systems*, In the proceeding of 5th International Conference on Computing and Artificial Intelligence (ICCAI 2019), ACM.

[CB9] **Nguyễn Tấn Cầm**, Nguyễn Văn Tâm, Văn Hồng Thư, Ung Văn Giàu, Phạm Văn Hậu, Nguyễn Anh Tuấn, 2018, "Đánh giá mối liên hệ giữa thông tin mô tả và quyền hạn của ứng dụng Android", *Tạp chí Thông tin và truyền thông, ISSN 1859-3550, Trang 108-113, Special Issues của SoIS2018, 2018.*

4.2.3 Giải pháp hữu ích liên quan đến luận án

Trong quá trình thực hiện luận án này, chúng tôi đã đăng ký 02 giải pháp hữu ích liên quan đến các nội dung chính của luận án. Cụ thể:

[SC1] "Hệ thống phân tích nguy cơ gây thất thoát thông tin nhạy cảm trong các thiết bị Android", *quyết định số 73924/QĐ-SHTT của Cục sở hữu trí tuệ, Bộ Khoa học và Công nghệ ngày 24/10/2017.*

[SC2] "Hệ thống phân tích ROM Android tùy biến để phát hiện rò rỉ thông tin nhạy cảm", *quyết định số 37370/QĐ-SHTT của Cục sở hữu trí tuệ, Bộ Khoa học và Công nghệ ngày 31/05/2018.*

4.2.4 Các đề tài đã tham gia

Trong quá trình thực hiện luận án, nghiên cứu sinh đã tham gia hai đề tài nghiên cứu khoa học. Trong đó, một đề tài thuộc Sở khoa học và Công nghệ, đề tài còn lại thuộc Đại học Quốc gia TPHCM. Cụ thể:

Tham gia đề tài "Xây dựng hệ thống phát hiện spyware trên nền tảng Android" với vai trò thành viên. Đây là đề tài cấp Sở khoa học và công nghệ Thành phố Hồ Chí Minh, thời gian thực hiện 2015-2017, đã được nghiệm thu với kết quả Khá.

Bên cạnh đó, nghiên cứu sinh tham gia đề tài "Một hướng tiếp cận trong việc đánh giá an toàn thông tin ROM Android" với vai trò thành viên. Đây là đề tài B, Đại học Quốc gia TPHCM, thời gian thực hiện 2016-2018, đã nghiệm thu với kết quả Tốt.

4.3 Hạn chế và hướng phát triển của luận án

4.3.1 Hạn chế

Phân tích các lớp thành phần (layer) của hệ điều hành Android một cách riêng biệt: Luận án phân tích thành phần ứng dụng cài sẵn và Application Framework một cách riêng biệt. Tuy nhiên, trong thực tế, có thể có các luồng dữ liệu bắt nguồn từ lớp thành phần này này và kết thúc ở lớp thành phần khác, hoặc luồng dữ liệu qua các lớp thành phần khác nhau để thực hiện việc gây thất thoát dữ liệu nhạy cảm. Do đó, tại thời điểm hiện tại, luận án chưa thể phát hiện các luồng dữ liệu liên lớp thành phần.

Một hạn chế nữa của luận án này là chưa có bộ dữ liệu thử nghiệm chuẩn đủ lớn để đánh giá phương pháp đề xuất và các nghiên cứu liên quan. Hiện tại, cộng đồng các nhóm nghiên cứu liên quan đến lĩnh vực nghiên cứu này chỉ sử dụng DroidBench. Do đó, việc mở rộng dataset này cũng là một cố gắng của nghiên cứu sinh trong quá trình thực hiện luận án này.

4.3.2 Hướng phát triển của luận án

Với giải pháp phân tích nguy cơ gây rò rỉ thông tin nhạy cảm trên nhiều ứng dụng trong luận án giúp phát hiện các trường hợp tấn công cộng tác có thể mở rộng trong tương lai để đánh giá, gom nhóm các ứng dụng nào trên chợ ứng dụng có mối liên hệ với nhau trong việc thực hiện tấn công cộng tác. Hiện chúng tôi phát triển thêm công cụ tải tự động các ứng dụng Android miễn phí từ Google Play. Trong tương lai, chúng tôi sẽ hoàn thiện công cụ này để tích hợp cùng hệ thống đề xuất trong luận án để trở thành một hệ thống xây dựng bản đồ nguy cơ bảo mật trên chợ ứng dụng. Bên cạnh đó, cần phát triển luận án theo hướng cho phép phát hiện các luồng dữ liệu liên lớp thành phần để giúp phân tích toàn diện hơn các tình huống gây thất thoát thông tin nhạy cảm trong các thiết bị sử dụng hệ điều hành Android.

Tài liệu tham khảo

[1] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, *et al.*, "FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," presented at the Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, Edinburgh, United Kingdom, 2014.

[2] D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel, "Composite Constant Propagation: Application to Android Inter-Component Communication Analysis," in *the 37th International Conference on Software Engineering (ICSE)*, 2015.

[3] S. Rasthofer, S. Arzt, and E. Bodden, "A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks," 2014.

[4] Google. (2019, Janruary 10). *Application security - Android Open Source Project*. Available: https://source.android.com/security/overview/app-security

[5] Gartner. (2018). *Gartner Says Worldwide Device Shipments Will Increase 2.1 Percent in 2018*. Available: https://www.gartner.com/en/newsroom/press-releases/2018-01-29-gartner-says-worldwide-device-shipments-will-increase-2-point-1-percent-in-2018

[6] IDC. (2018). *Smartphone Market Share*. Available: https://www.idc.com/promo/smartphone-market-share/os

[7] Semantec. (2015, May). *2015 Internet Security Threat Report, Volume 20* Available: http://www.symantec.com/security_response/publications/threatreport.jsp

[8] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," presented at the Proceedings of the 16th ACM conference on Computer and communications security, Chicago, Illinois, USA, 2009.

[9] T. Luo, J. Wu, M. Yang, S. Zhao, Y. Wu, and Y. Wang, "MAD-API: Detection, Correction and Explanation of API Misuses in Distributed Android Applications," in *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, 2018, pp. 123-140.

[10] F. Wei, S. Roy, X. Ou, and Robby, "Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps," *ACM Trans. Priv. Secur.*, vol. 21, pp. 1-32, 2018.

[11] S. Liang, A. W. Keep, M. Might, S. Lyde, T. Gilray, P. Aldous, *et al.*, "Sound and precise malware analysis for android via pushdown reachability and entry-point saturation," in *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, 2013, pp. 21-32.

[12] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale," in *International Conference on Trust and Trustworthy Computing*, 2012, pp. 291-307.

[13] M. Xia, L. Gong, Y. Lyu, Z. Qi, and X. Liu, "Effective real-time android application auditing," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 899-914.

[14] D. Titze and J. Schütte, "Apparecium: Revealing data flows in android applications," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015, pp. 579-586.
[15] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, 2015, pp. 303-313.
[16] M. Zhang and H. Yin, "Automatic Generation of Vulnerability-Specific Patches for Preventing Component Hijacking Attacks," in *Android Application Security*, ed: Springer, 2016, pp. 45-61.
[17] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: semantics-based detection of Android malware through static analysis," presented at the Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, China, 2014.
[18] V. Rastogi, Z. Qu, J. McClurg, Y. Cao, and Y. Chen, "Uranine: Real-time privacy leakage monitoring without system modification for android," in *International Conference on Security and Privacy in Communication Systems*, 2015, pp. 256-276.
[19] Y. Zhou, L. Wu, Z. Wang, and X. Jiang, "Harvesting developer credentials in android apps," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2015, p. 23.
[20] W. Huang, Y. Dong, A. Milanova, and J. Dolby, "Scalable and precise taint analysis for android," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 106-117.
[21] Z. Wu, X. Chen, Z. Yang, and X. Du, "Reducing Security Risks of Suspicious Data and Codes through a Novel Dynamic Defense Model," *IEEE Transactions on Information Forensics and Security*, 2019.
[22] D. Rathi and R. Jindal, "DroidMark: A Tool for Android Malware Detection using Taint Analysis and Bayesian Network," *arXiv preprint arXiv:1805.06620*, 2018.
[23] Z. Meng, Y. Xiong, W. Huang, L. Qin, X. Jin, and H. Yan, "AppScalpel: Combining static analysis and outlier detection to identify and prune undesirable usage of sensitive data in Android applications," *Neurocomputing*, vol. 341, pp. 10-25, 2019.
[24] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for android apps," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 518-529.
[25] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," presented at the Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis, Edinburgh, United Kingdom, 2014.
[26] Y. Zhang, Y. Li, T. Tan, and J. Xue, "Ripple: Reflection analysis for android apps in incomplete information environments," *Software: Practice and Experience*, vol. 48, pp. 1419-1437, 2018.
[27] Y. Tsutano, S. Bachala, W. Srisa-An, G. Rothermel, and J. Dinh, "An efficient, robust, and scalable approach for analyzing interacting android apps," in *Proceedings of the 39th International Conference on Software Engineering*, 2017, pp. 324-334.
[28] L. Li, A. Bartel, T. Bissyandé, J. Klein, and Y. Traon, "ApkCombiner: Combining Multiple Android Apps to Support Inter-App Analysis," in *ICT Systems Security and Privacy Protection*. vol. 455, H. Federrath and D. Gollmann, Eds., ed: Springer International Publishing, 2015, pp. 513-527.
[29] L. Li, A. Bartel, T. Bissyande, J. Klein, Y. L. Traon, S. Arzt, *et al.*, "IccTA: Detecting Inter-Component Privacy Leaks in Android Apps," presented at the The 37th International Conference on Software Engineering (ICSE), Firenze, Italy, 2015.
[30] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," *computers & security*, vol. 65, pp. 121-134, 2017.
[31] Z. Meng, Y. Xiong, W. Huang, F. Miao, T. Jung, and J. Huang, "Divide and Conquer: Recovering Contextual Information of Behaviors in Android Apps around Limited-quantity Audit Logs," *arXiv preprint arXiv:1809.07036*, 2018.
[32] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, *et al.*, "SmartDroid: an automatic system for revealing UI-based trigger conditions in android applications," presented at the Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices, Raleigh, North Carolina, USA, 2012.
[33] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini, and E. De Cristofaro, "A Family of Droids: Analyzing Behavioral Model based Android Malware Detection via Static and Dynamic Analysis," *arXiv preprint arXiv:1803.03448*, 2018.
[34] M. Hammad, H. Bagheri, and S. Malek, "DelDroid: An automated approach for determination and enforcement of least-privilege architecture in android," *Journal of Systems and Software*, vol. 149, pp. 83-100, 2019.
[35] M. Zheng, M. Sun, and J. C. S. Lui, "DroidRay: a security evaluation system for customized android firmwares," presented at the Proceedings of the 9th ACM symposium on Information, computer and communications security, Kyoto, Japan, 2014.
[36] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic Detection of Capability Leaks in Stock Android Smartphones," in *The 19th Annual Network & Distributed System Security Symposium*, 2012.
[37] Y. Aafer, X. Zhang, and W. Du, "Harvesting Inconsistent Security Configurations in Custom Android ROMs via Differential Analysis," presented at the USENIX SECURITY, 2016.
[38] X. Zhou, Y. Lee, N. Zhang, M. Naveed, and X. Wang, "The Peril of Fragmentation: Security Hazards in Android Device Driver Customizations," presented at the Proceedings of the 2014 IEEE Symposium on Security and Privacy, 2014.
[39] Y. Cao, Y. Fratantonio, A. Bianchi, M. Egele, C. Kruegel, G. Vigna, *et al.*, "EdgeMiner: Automatically Detecting Implicit Control Flow Transitions through the Android Framework," in *NDSS*, 2015.

[40] M. Backes, S. Bugiel, E. Derr, P. D. McDaniel, D. Octeau, and S. Weisgerber, "On Demystifying the Android Application Framework: Re-Visiting Android Permission Specification Analysis," in *USENIX Security Symposium*, 2016, pp. 1101-1118.
[41] Y. Cao, Y. Fratantonio, A. Bianchi, M. Egele, C. Kruegel, G. Vigna, *et al.*, "EdgeMiner: Automatically Detecting Implicit Control Flow Transitions through the Android Framework," in *NDSS*.
[42] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, "The impact of vendor customizations on android security," presented at the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, Berlin, Germany, 2013.
[43] F. Liu, H. Cai, G. Wang, D. D. Yao, K. O. Elish, and B. G. Ryder, "MR-Droid: A scalable and prioritized analysis of inter-app communication risks," *Proc. of MoST*, 2017.
[44] E. SPRIDE. (2018, March 10). *DroidBench – Benchmarks*. Available: http://sseblog.ec-spride.de/tools/droidbench/
[45] L. Li, A. Bartel, J. Klein, and Y. le Traon, "Automatically Exploiting Potential Component Leaks in Android Applications," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, 2014, pp. 388-397.
[46] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," presented at the Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, Chicago, Illinois, USA, 2011.
[47] T. Ravitch, E. R. Creswick, A. Tomb, A. Foltzer, T. Elliott, and L. Casburn, "Multi-App Security Analysis with FUSE: Statically Detecting Android App Collusion," presented at the Proceedings of the 4th Program Protection and Reverse Engineering Workshop, New Orleans, LA, USA, 2014.
[48] A. Bosu, F. Liu, D. Yao, and G. Wang, "Collusive Data Leak and More: Large-scale Threat Analysis of Inter-app Communications," presented at the Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2017.
[49] S. Bhandari, V. Laxmi, A. Zemmari, and M. S. Gaur, "Intersection automata based model for android application collusion," in *Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on*, 2016, pp. 901-908.
[50] S. Bhandari, F. Herbreteau, V. Laxmi, A. Zemmari, P. S. Roop, and M. S. Gaur, "Poster: Detecting inter-app information leakage paths," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 908-910.
[51] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," presented at the Proceedings of the 9th international conference on Mobile systems, applications, and services, Bethesda, Maryland, USA, 2011.
[52] D. Boxler and K. R. Walcott, "Static Taint Analysis Tools to Detect Information Flows," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, 2018, pp. 46-52.
[53] Z. Bohluli and H. R. Shahriari, "Detecting Privacy Leaks in Android Apps using Inter-Component Information Flow Control Analysis," in *2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, 2018, pp. 1-6.
[54] G. Bai, Q. Ye, Y. Wu, H. Botha, J. Sun, Y. Liu, *et al.*, "Towards model checking android applications," *IEEE Transactions on Software Engineering*, vol. 44, pp. 595-612, 2018.
[55] H. Cao, J. Jiao, and D. Li, "A Static Analysis Model for Implicit Information Leakage in Android Application," in *2018 IEEE 18th International Conference on Communication Technology (ICCT)*, 2018, pp. 1133-1140.
[56] D. D. Perez and W. Le, "Specifying Callback Control Flow of Mobile Apps Using Finite Automata," *IEEE Transactions on Software Engineering*, 2019.
[57] A. Tiwari, S. Groß, and C. Hammer, "IIFA: Modular Inter-app Intent Information Flow Analysis of Android Applications," *arXiv preprint arXiv:1812.05380*, 2018.
[58] F. Sattler, A. von Rhein, T. Berger, N. S. Johansson, M. M. Hardø, and S. Apel, "Lifting inter-app data-flow analysis to large app sets," *Automated Software Engineering*, vol. 25, pp. 315-346, 2018.
[59] A. I. Ali-Gombe, B. Saltaformaggio, D. Xu, and G. G. Richard III, "Toward a more dependable hybrid analysis of android malware using aspect-oriented programming," *computers & security*, vol. 73, pp. 235-248, 2018.
[60] G. Meng, R. Feng, G. Bai, K. Chen, and Y. Liu, "DroidEcho: an in-depth dissection of malicious behaviors in Android applications," *Cybersecurity*, vol. 1, p. 4, 2018.
[61] Google. (2017, March 10). *Android Open Source Project*. Available: https://source.android.com/
[62] A. K. Jha and W. J. Lee, "An empirical study of collaborative model and its security risk in Android," *Journal of Systems and Software*, vol. 137, pp. 550-562, 2018.
[63] F. Liu, H. Cai, G. Wang, D. Yao, K. O. Elish, and B. G. Ryder, "Prioritized Analysis of Inter-App Communication Risks," presented at the Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, Arizona, USA, 2017.
[64] Xposed. (2016). *Xposed framework*. Available: http://repo.xposed.info/
[65] D. T. Milano. (2017, April 2). *Android ViewServer client*. Available: https://github.com/dtmilano/AndroidViewClient
[66] L. Xue, C. Qian, H. Zhou, X. Luo, Y. Zhou, Y. Shao, *et al.*, "NDroid: Toward Tracking Information Flows Across Multiple Android Contexts," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 814-828, 2019.
[67] (2017, 5/7/2017). *Xposed framework*. Available: http://repo.xposed.info/module/de.robv.android.xposed.installer
[68] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of android apps," *SIGPLAN Not.*, vol. 48, pp. 641-660, 2013.