ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN NAM CHUNG

THEO DÕI CÁC NGUỒN DỮ LIỆU NHẠY CẢM TRÊN CÁC THIẾT BỊ DI ĐỘNG CHẠY HỆ ĐIỀU HÀNH ANDROID

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội – 2017

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN NAM CHUNG

THEO DÕI CÁC NGUỒN DỮ LIỆU NHẠY CẢM TRÊN CÁC THIẾT BỊ DI ĐỘNG CHẠY HỆ ĐIỀU HÀNH ANDROID

Ngành: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm
Mã số: 60480103

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS. Nguyễn Đại Thọ

Hà Nội, 2017

Lời Cảm Ơn

Trong quá trình học tập cũng như nghiên cứu để hoàn thành đề tài luận văn thạc sĩ một cách hoàn chỉnh, bên cạnh sự nỗ lực của bản thân còn có sự hướng dẫn nhiệt tình của quý Thầy Cô, cũng như sự động viên, hỗ trợ của gia đình và đồng nghiệp trong suốt thời gian học tập nghiên cứu và thực hiện luận văn thạc sĩ.

Tôi xin chân thành gửi lời cảm ơn đến các quý Thầy Cô, các cán bộ nhân viên thuộc Trường Đại học Công nghệ, ĐHQGHN đã trang bị kiến thức, tạo môi trường điều kiện thuận lợi nhất cho tôi trong suốt quá trình học tập và thực hiện luận văn này.

Với lòng kính trọng và biết ơn, tôi xin được bày tỏ lời cảm ơn sâu sắc đến tiến sĩ Nguyễn Đại Thọ, giảng viên khoa CNTT - Trường Đại học Công nghệ, ĐHQGHN đã trực tiếp hướng dẫn tận tình cho tôi trong suốt quá trình thực hiện nghiên cứu này.

Cuối cùng tôi xin chân thành cảm ơn đến gia đình, các đồng nghiệp đã hỗ trợ cho tôi rất nhiều trong quá trình học tập, nghiên cứu và thực hiện đề tài luận văn thạc sĩ một cách hoành chỉnh.

Hà Nội, tháng 6 năm 2017

Học viên thực hiện

Nguyễn Nam Chung

Lời Cam Đoan

Tôi xin cam đoan đây là công trình nghiên cứu khoa học độc lập của riêng tôi. Các kết quả nghiên cứu trong luận văn do tôi tự tìm hiểu, phân tích một cách trung thực, khách quan. Các thông tin trích dẫn trong luận văn đã được chỉ rõ nguồn gốc rõ ràng và được phép công bố.

Nếu có bất kỳ điều gì không đúng như đã nêu trên, tôi xin hoàn toàn chịu trách nhiệm về luân văn của mình.

Tác giả

Nguyễn Nam Chung

MỤC LỤC

Danh mục các ký hiệu và chữ viết tắt

STT	Ký hiệu/viết tắt	Ý nghĩa
1	CFG	Control Flow Graph: Biểu đồ luồng điều khiển
2	DEX	Định dạng mã code Dalvik Executable được thiết kết cho các hệ thống hạn chế về dung lượng bộ nhớ và tốc độ bộ vi xử lý. Các chương trình việt bằng ngôn ngữ Java thường được dịch ra mã chạy cho máy ảo Java, nó được chuyển sang mã Dalvik và lưu trữ dưới dạng tệp DEX
3	GPU	Graphics Processing Unit: Đơn vị xử lý đồ họa (), thường sử dụng cho các hệ thống nhúng, điện thoại di động, máy tính cá nhân và các thiết bị game. Nó hỗ trợ hiệu quả trong đồ họa máy tính và xử lý ảnh
4	HĐH	Hệ điều hành, hệ thống phần mềm quản lý tài nguyên phân cứng và phần mềm của thiết bị
5	IPC	Inter-Processing Communication: Kết nối liên xử lý
6	JVM	Java Virtual Machine: Máy ảo Java
7	Native code	Mã nguyên gốc, như các mã của trình biên dịch, mã máy
8	Native method	Phương thức nguyên gốc, như các hàm nguyên thủy của 1 hệ thống, hệ điều hành do nhà phát triển hệ thống viết sẵn
9	Taint	Thông tin kiểm tra được định danh
10	TBDĐ	Thiết bị di động
11	VM	Virtual Machine: Một hệ thống máy tính giả lập dựa trên các kiến trúc máy tính và cũng cấp các chức năng như một máy tính vật lý. Nó được cài đặt trên máy tính thật thông qua các chương trình quản lý máy ảo như HyperV, VMware, VirtualBox, Virtual PC …

MỞ ĐẦU

Các thiết bị di động (TBDĐ) hiện nay có rất nhiều các cảm biến và dịch vụ bên trong không gian riêng tư của người dùng. Vậy nên chúng có khả năng giám sát rất nhiều khía cạnh nhạy cảm trong cuộc sống người sử dụng (ví dụ: vị trí, sức khỏe hay giao thiệp). Người dùng thường giao dịch thanh toán trực tuyến, trong nhiều tình huống họ không đánh giá đầy đủ về bản chất cũng như mức độ thông tin sẽ được khaithác bởi các ứng dụng bên thứ 3. Sự gia tăng nhanh chóng của các TBDĐ khiến chúng trở thành không thể thiếu với cuộc sống của nhiều người. Thật vậy, các thiết bị cung cấp cho người dùng một loạt các dịch vụ thiết yếu (ví dụ: định vị, liên lạc và kết nối Internet) cùng các chức năng hữu ích (ví dụ: nghe nhạc, nhiếp ảnh, xem truyền hình, mua sắm trực tuyến). Để đáp ứng các dịch vụ này, các TBDĐ hiện đại đã được trang bị rất nhiều cảm biến, khả năng thu thập thông tin và môi trường xung quanh về người dùng vô cùng phong phú. Người dùng và nhà phát triển cũng đã chấp nhận các công nghệ thu thập thông tin để đổi lấy rất nhiều tính năng công nghệ cao mà chúng mang lại. Trên thực tế, rất nhiều ứng dụng mời chào các dịch vụ hoàn toàn miễn phí với sự đánh đổi ẩn dấu về việc thu thập dữ liệu mà hầu hết sẽ được dùng cho việc quảng cáo.

Có rất nhiều nghiên cứu đã chỉ ra, người dùng thường hành động mà không hiểu mức độ thông tin của họ có thể được trích ra từ các thông tin được thu thập này. Ví dụ trong một nghiên cứu gần đây về hành vi quảng cáo đã chỉ ra rằng người dùng đã không hiểu cách quảng cáo hoạt động và tìm kiếm thông tin mà các nhà quảng cáo có về họ là đáng sợ và rùng mình. Ngay cả khi người dùng ý thức việc thu thập dữ liệu, họ không thể hoàn toàn nhận ra những hàm ý không trực quan về việc chia sẻ dữ liệu của họ. Các nhà nghiên cứu đã chỉ ra các cảm biến trên các thiết bị có thể được dùng bí mật để bắt các phím bấm, chạm trên điện thoại để dò tìm vị trí, ghi âm giọng nói hay các hoạt động thường ngày của người dùng. Phần lớn các thiết bị được chạy trên hệ điều hành (HĐH) Android [1] và iOS[2]. Cùng với sự bùng nổ của thị trường TBDĐ sử dụng HĐH nguồn mở thì Android đã trở thành HĐH phổ biến nhất hiện nay với kho ứng dụng lên đến hơn 2,5 triệu ứng dụng. Có thể dự đoán, có hàng triệu người dùng đang mang theo các công nghệ thu thập dữ liệu tập trung về quyền riêng tư. Dữ liệu thu thập được có thể được truy xuất bởi rất nhiều bên và thường không có sự cho phép rõ

[1] Android: Hệ điều hành cho TBDĐ mã nguồn mở của Google, website chính thức tại địa chỉ https://www.android.com

[2] iOS: Hệ điều hành cho TBDĐ độc quyền của Apple, website chính thức tại địa chỉ https://www.apple.com/ios

ràng của người dùng. Thế nên việc giám sát các luồng dữ liệu do các ứng dụng trao đổi với bên ngoài là hết sức cần thiết. Để đảm bảo tính bảo mật, toàn vẹn và khả dụng của thông tin mà TBDĐ truy cập, lưu trữ và xử lý là một thách thức khó khăn. Điều này đặc biệt đúng trong bối cảnh ngày nay môi trường di động đang phát triển với các loại thông tin phức tạp và phần mềm phải cùng tồn tại một cách an toàn trên cùng một thiết bị để giữa an toàn dữ liệu. Nhưng hiện nay việc kiểm soát an ninh vẫn chưa theo kịp với những rủi ro gây ra bởi các TBDĐ. Các nhà phát triển cũng như các doanh nghiệp vẫn đang nỗ lực đưa ra các giải pháp an ninh thông tin trên các thiết bị động và hiện nay có một số hệ nổi bật về cả phần mềm, phần cứng cũng như tích hợp như: Samsung Knox, BlackBerry Balance, AndroidLeaks, SCANDAL[1], IccTAp[2], TaintDroid,… và dưới đây là những phân tích sơ bộ về ưu nhược điểm của các hệ thống này, chi tiết sẽ được miêu tả ở phần các hệ thống bảo mật trong chương 1.

- Samsung Knox là một nền tảng an ninh cho TBDĐ về cả phần cứng lẫn phần mềm. Nó cung cấp các tính năng bảo mật cho cả dữ liệu cá nhân cũng như doanh nghiệp trên cùng một TBDĐ. Có khả năng bảo mật đa lớp và phần vùng an toàn. Nhưng để sử dụng Knox, thiết bị phải được hỗ trợ công nghệ ảo hóa ở mức phần cứng và hiện nay chỉ giới hạn ở các thiết bị của Samsung.
- BlackBerry Balance công cụ cho phép quản lý dữ liệu cá nhân chuyên biệt, độc lập trên TBDĐ. Nó cũng có khả năng phân vùng an toàn, nhưng ưu điểm nổi bật là khả năng mã hóa dữ liệu ở mức cao với hệ thống khóa phức tạp. Tuy nhiên hiện nay nó chỉ hỗ trợ cho các thiết bị chạy HĐH Blackberry và chỉ từ phiên bản 10 trở lên.
- SCANDAL là một bộ phân tích tĩnh có thể phát hiện rò rỉ thông tin cá nhân trong các ứng dụng chạy HĐH Android. Nhược điểm là trong quá trình xử lý nó có thể mất nhiều thời gian cũng như không gian nhớ của thiết bị.
- IccTA là một công cụ mã nguồn mở phát hiện rò rỉ thông tin. Nó dùng chuyển mã Dalvik[3] và kết nối trực tiếp đến các thành phần để xử lý luồng dữ liệu giữa giữa chúng. Nhưng hiện tại nó chỉ xử lý hiệu quả được các lời gọi với các tham số là chuỗi cố định.
- AndroidLeaks là một hệ thống thực hiện xử lý luồng dữ liệu để xác định xem thông tin cá nhân có bị truy xuất không. Hệ thống còn một vài giới hạn về độ chính xác và sai số và không hỗ trợ xử lý luồng điều.

TaintDroid là một hệ thống có khả năng kiểm tra truy cập thông tin nhạy

[1] SCANDAL: Static Analyzer for Detecting Privacy Leaks in Android Applications
[2] IccTA: Detecting Inter-Component Privacy Leaks in Android Apps
[3] Dalvik: Xây dựng riêng cho máy ảo, đặc biệt cho phát triển Android

cảm của người dùng ở mức thời gian thực. Nó kiểm tra luồng thông tin trên các TBDĐ chạy HĐH Android rất hiệu quả. Tuy còn hạn chế khi không kiểm tra được dưới dạng luồng điều khiển, nhưng đây không phải định hướng mức kiến trúc khi ban đầu khi xây dựng. TaintDroid có thể chạy trên các thiết bị chạy HĐH Android từ phiên bản 2.1 trở đi. Nó hỗ trợ kiểm tra dấu vết (taint[1]) thông tin cá nhân nhạy cảm của người dùng bị các ứng dụng truy cập. Mục tiêu chính của hệ thống là rò tìm khi nào dữ liệu nhạy cảm bị gửi đi từ những ứng dụng không tin cậy. TaintDroid hiện tại có thể kiểm tra nhiều loại thông tin như: vị trí, số điện thoại, máy ảnh, số IMEI, lịch sử trình duyệt. Chính vì các ưu điểm nổi bật của nó so với các hệ thống khác về theo dõi truy cập thông tin nhạy cảm mà nó được chọn làm đề tài nghiên cứu và cải tiến trong khuôn khổ luận văn. Tuy hệ thống đã có khả năng kiểm soát và cảnh báo truy cập trái phép các nhóm thông tin kể trên, nhưng chỉ cảnh báo chung ở mức loại thông tin theo taint. Việc này chỉ hỗ trợ người dùng kiểm soát chung nhất việc truy cập thông tin nhạy cảm mà chưa biết được chính xác những thông tin gì trong đó bị truy cập và gửi đi trái phép.

Hướng cải tiến trong khuôn khổ luận văn là bổ sung cho hệ thống TaintDroid tính năng cảnh báo người dùng khi ứng dụng truy cập thông tin nhạy cảm trong lịch sử trình duyệt. Mục tiêu chính của cải tiến là sẽ thông báo cho người dùng khi có ứng dụng không tin cậy truy cập đến tên đăng nhập, mật khẩu hay mã số thẻ tín dụng. Với nhu cầu và thói quen của người dùng điện thoại thông minh (Smartphone) hiện nay, việc truy cập Internet là thường xuyên và cùng với đó là việc sử dụng các ứng dụng truy Internet như FaceBook, Twitter, Chrome, … để làm việc và giải trí. Như chúng ta thấy, với cơ chế hoạt động của trình duyệt web cũng như các ứng dụng hoạt động trên nền web thì có rất nhiều thông tin được trình duyệt lưu lại trong quá trình sử dụng. Các thông tin đó được lưu lại thành dữ liệu lịch sử của trình duyệt (browser history), trong đó chứa nhiều thông tin nhạy cảm.

Cải tiến đã hoàn thành với kết quả đạt được cụ thể và rõ ràng như định hướng đề ra. Hệ thống đã có thể kiểm tra được chính xác các taint chứa thông tin nhạy cảm trong lịch sử trình duyệt bị các ứng dụng không tin cậy truy cập. Việc cải tiến cũng không làm ảnh hưởng đến các luồng xử lý cũng như hiệu năng của hệ thống hiện tại. Cụ thể khi thông tin về tên truy cập, mật khẩu hay mã số thẻ tín dụng bị truy cập, hệ thống sẽ hiện thông báo riêng so với các thông báo sẵn có bằng hình thức thay đổi đèn LED và tần suất nhấp nháy. Việc cải tiến được

[1] Taint: Thẻ thông tin được truyền giữa HĐH và ứng dụng thời gian thực.

kiểm chứng trên môi trường thật, thiết bị được sử dụng là điện thoại di động Google Nexus 4. Việc xây dựng, cài đặt cũng như chạy thử đều tuân thủ các bước do nhóm phát triển hệ thống TaintDroid đưa ra. Toàn bộ tài liệu luận văn được bố trí với bố cục các chương mục tóm tắt như sau:

- Chương 1 - Bảo mật riêng tư trên các thiết bị di động: Chương này giới thiệu các khái niệm, tầm quan trọng cũng như các phương pháp và nguyên lý bảo mật riêng tư. Ngoài ra còn giới thiệu chi tiết về bảo mật cho trình duyệt web và một số hệ thống an ninh cho TBDĐ tiêu biểu.
- Chương 2 - Hệ thống TantDroid: Giới thiệu từ tổng quan đến chi tiết các thành phần của TaintDroid cũng như phân tích đánh giá hiệu năng của hệ thống.
- Chương 3 - Cải tiến theo dõi nguồn dữ liệu nhạy cảm: Chương này miêu tả chi tiết việc cải tiến hệ thống TaintDroid, từ giải pháp chi tiết về kiểm soát truy cập taint lịch sử trình duyệt web đến giải pháp tổng thể để kiểm soát truy cập các loại taint mà hệ thống hiện có. Ngoài ra cũng nêu nên những vấn đề thực tiễn trong quá trình cải tiến và đóng góp trong cải tiến cho hệ thống.
- Chương 4 - Kết quả thử nghiệm: Miêu tả chi tiết quá trình thực nghiệm từ môi trường, thiết bị đến việc chạy ứng dụng trên thiết bị thật. Đưa ra các đánh giá về cải tiến theo các tiêu chí cụ thể, hướng thảo luận và định hướng tiếp theo.
- Kết luận: Chỉ ra tính ứng dụng của hệ thống cải tiến, các hạn chế còn tồn tại.
- Tài liệu thao khảo: Liệt kê đầy đủ danh sách các tài liệu tham khảo trong quá trình thực hiện luận văn.
- Phụ lục: Hướng dẫn chi tiết việc xây dựng hệ thống từ việc cài đặt, cấu hình HĐH cho đến cách lấy mã nguồn. Cuối phần là toàn bộ mã nguồn của giải pháp cải tiến chi tiết và giải pháp tổng quát.

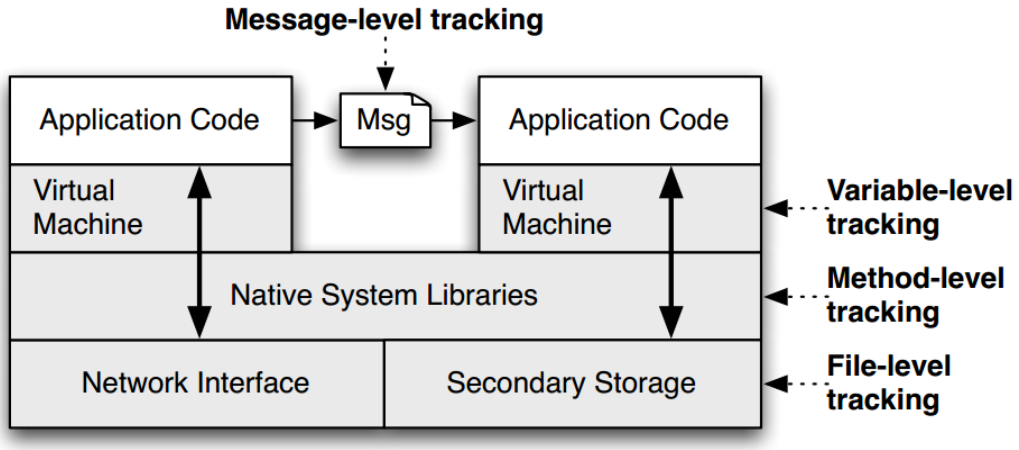
Chương 1. Bảo mật riêng tư trên các thiết bị di động

1.1. Bối cảnh chung

Trong xã hội ngày nay, nói đến smartphone là nói đến công nghệ mới. Các smartphone về cơ bản luôn kết nối nhiều dữ liệu cá nhân trong cuộc sống, không chỉ dữ liệu đơn thuần như danh bạ mà nhiều loại dữ liệu kiểu mới như vị trí, sở thích mua sắm trực tuyến. Chúng cũng có khả năng tải và chạy các ứng dụng của bên thứ 3 có kết nối với Internet. Một ví dụ điển hình về ứng dụng hình nền (Wallpaper) gửi thông tin số điện thoại tới nhà phát triển. Khi một ứng dụng chạy thường có thể truy cập bất cứ thông tin nào trên thiết bị không tường minh, ngay cả cách chúng thực hiện việc này. Trong nghiên cứu, nhóm tác giả đã chọn tên Dynamic Taint Analysis, thỉnh thoảng gọi là Taint Tracking. Ý tưởng cơ bản là đánh dấu thẻ taint thông tin nhạy cảm tại nguồn và sau đó theo dấu dữ liệu được chuyển đi qua hệ thống. Trong ngữ cảnh của báo cáo, dữ liệu được đánh dấu truyền qua giao diện mạng của smartphone khi đó có thể biết thông tin có nhạy cảm hay không?

Có rất nhiều khó khăn khi xây dựng một hệ thống như vậy trên smartphone . Mục đích của thiết kế là tạo ra một hệ thống gọn nhẹ, tối ưu về bộ nhớ và kiểm tra ở mức thực thi (real-time), chạy trực tiếp trên thiết bị thật với các ứng dụng thật và dưới đây là một số khó khăn khi xây dựng:

- Smartphone rất hạn chế về tài nguyên, bộ nhớ bị giới hạn và mọi xử lý được thực hiện đều tiêu tốn pin. Nếu hệ thống kiểm tra real-time, phải được cân nhắc việc người dùng và hệ thống cùng sử dụng nên nó phải thực sự gọn nhẹ. Ứng dụng bên thứ 3 dưới dạng mã biên dịch nên không thể phân tích mã nguồn. Chúng không chỉ đơn giản đọc một loại dữ liệu và gửi đi ngay mà sẽ dùng dữ liệu đó bằng một số cách hoặc kết hợp với dữ liệu khác trước khi gửi đi.
- Các ứng dụng có thể chia sẻ thông tin với các ứng dụng khác, có nghĩa việc kiểm tra cũng phải xử lý chéo nhiều tiến trình.
- Việc kiểm tra phải thực hiện trên thiết bị thật. Với một hệ thống giả lập thì việc điều khiển trên phần cứng và bộ nhớ ảo có thể xác định được mọi thứ mà một ứng dụng làm. Nhưng trên một thiết bị thật, làm cách nào để có thể biết mọi việc ứng dụng làm ở mức đủ thấp trên hệ thống? Trong quá trình nghiên cứu, nhóm tác giả đã đưa ra mô hình sơ lược hướng tiếp cận đa mức về kiểm tra hiệu năng smartphone như sau:



Hình 1.1. Mô hình kiểm tra hiệu năng điện thoại di động

1.2. Khái niệm bảo mật riêng tư

Chính sách riêng tư là một tuyên bố pháp lý xác định chủ quyền thương mại với dữ liệu cá nhân sẽ được thu thập bởi người sử dụng, bao gồm cả việc dữ liệu được xử lý như thế nào vào ra sao. Từ những năm 1960, Hội đồng châu Âu đã nghiên cứu về sự mở rộng của Internet và tập trung vào sự ảnh hưởng của công nghệ đến các quyền con người. Nghiên cứu về các chính sách để bảo vệ dữ liệu cá nhân. Nó được chúng ta biết đến với tên gọi tiếng Anh "Privacy Policy". Tên gọi này chỉ ra thỏa thuận pháp lý về quyền và sự bảo vệ dữ liệu cá nhân một cách đầy đủ, thỏa thuận này cũng có thể được biết tới dưới các tên sau:

- Cam kết riêng tư
- Thông báo riêng tư
- Thông tin riêng tư
- Trang riêng tư
- Chính sách bảo mật có thể được dùng cho cả các ứng dụng web hay di động nếu nó bao gồm các nền tảng có hoạt động thương mại bên trong. Các nội dung của chính sách bảo mật có thể khác nhau tùy vào từng quốc gia, phụ thuộc vào luật pháp của từng nước nhưng hầu hết đều tuân thủ các điểm mấu chốt đảm bảo các hoạt động thương mại khi đối xử với dữ liệu cá nhân sau:
- Thông báo: Các dữ liệu được thu thập (bởi cá nhân hay tổ chức) phải minh bạch trong việc sử dụng thông tin cá nhân trước khi được thu thập.
- Lựu chọn: Các công ty thu thập dữ liệu phải tôn trọng các lựa chọn của người dùng về những thông tin cá nhân cung cấp ra.
- Truy xuất: Người dùng cũng có thể xem hay kiểm tra tính xác thực của dữ liệu cá nhân được thu thập bởi các công ty.
- Bảo mật: Các công ty có trách nhiệm hoàn toàn về tính chính xác và bảo mật (giữ nguyên tác quyền) của thông tin cá nhân được thu tập.

1.3. Tầm quan trọng của bảo mật riêng tư

Chính sách bảo mật là một trong những tài liệu quan trọng nhất của bất kỳ một ứng dụng web hay di động. Nó miêu tả chi tiết các quan điểm và thủ tục trong việc thu thập thông tin từ người sử dụng. Các phần chính của một tài liệu chính sách bảo mật được miêu tả dưới đây.

- Giới thiệu: Là phần miêu tả cho người đọc biết về tổ chức, người sử hữu và bất cứ thông tin hay chức năng đặc biệt nào mà ứng dụng có. Nếu ứng dụng có các điều kiện đặc biệt cho việc thu thập thông tin từ trẻ em (dưới 16 tuổi, vv) thì phải thông báo một cách rõ ràng và in đậm phần này.
- Thu thập thông tin: Người dùng có quyền biết những thông tin gì được thu thập. Phải tường minh trong việc thu thập chi tiết bằng việc hoàn thành các mẫu hỏi đáp.
- Phương pháp thu thập: Phần này miêu tả chi tiết các phương pháp được dùng để thu thập thông tin, có được tự động hóa không? có được thu thập với việc người dùng điền vào các mẫu không? Tất cả các câu hỏi sẽ giúp cho việc tạo ra một đặc tả chi tiết về cách thu thập thông tin.
- Lưu trữ thông tin: Làm thế nào để lưu trữ thông tin? Nếu lưu trữ thông tin trong cơ sở dữ liệu và vùng lưu trữ ở một quốc gia nào thì có thể phải được đăng ký với tổ chức an ninh dữ liệu của quốc gia đó. Các máy chủ đặt ở quốc gia nào thì phải được quốc gia đó cấp phép truyền tải dữ liệu ra bên ngoài. Người dùng có quyền biết việc thông tin của họ được lưu trữ trong môi trường an toàn và bảo mật.
- Thông tin liên hệ: Nó rất quan trọng để đảm bảo tính minh bạch nhất có thể và cho phép người dùng liên hệ với người sở hữu ứng dụng nếu họ cần. Ứng dụng cũng phải hỗ trợ liên hệ qua cả thư điện tử cũng như địa chỉ cố định.

1.4. Các phương pháp và công cụ đảm bảo tính riêng tư

Chính sách với người dùng ứng dụng đặc biệt quan trọng, tài liệu chính sách bảo mật chỉ ra các phương thức cho việc nhận và thu thập thông tin cá nhân bởi ứng dụng cũng như cách dùng chúng. Dưới đây là các phương pháp và công cụ điển hình trong việc đảm bảo tính riêng tư:

- Lưu tệp (Log tệp): Thông tin bên trong tệp bao gồm các địa chỉ Internet, kiểu trình duyệt, nhà cung cấp dịch vụ Internet, ngày tháng, các thông tin nhân khẩu học.
- Cookie: Lưu trữ thông tin về sở thích người dùng, các bản ghi thông tin đặc thù của người dùng khi truy cập vào trang web, các nội dung tùy biến dựa trên kiểu trình duyệt, kiểu ứng dụng hoặc các thông tin người dùng gửi qua trình duyệt/ứng dụng.

- DoubleClick DART[1] Cookie: Là một công cụ của Google sử dụng cookie để phục vụ quảng cáo trên các ứng dụng. Nó được Google sử dụng để quảng cáo tới người dùng dựa trên những lần truy cập ứng dụng. Người dùng có thể không dùng DART cookie bằng việc vào trang "chính sách riêng tư"[2] của Google để xác nhận.
- Các Cookie quảng cáo khác: Các máy chủ quảng cáo của bên thứ 3 và các mạng quảng cáo dùng công nghệ để đưa ra các quảng cáo, liên kết trên ứng dụng và gửi trực tiếp đến các trình duyệt của người dùng.
- Dữ liệu vùng: Người dùng có thể đưa thông tin về vùng như mã bưu điện hay địa chỉ, chúng sẽ được lưu trữ bởi các dịch vụ vùng. Các dữ liệu định vị có thể được lưu trữ để cung cấp các tính năng định vị toàn cầu, kết nối người dùng theo vùng.
- Google Analytics: Là một công cụ phân tích dùng cookie dưới dạng các tệp văn bản trên thiết bị để giúp các ứng dụng phân tích cách người dùng sử dụng ứng dụng đó. Nếu người dùng muốn vô hiệu hóa cookie, họ có thể sử dụng tùy chọn của trình duyệt.

1.5. Các nguyên lý chung đảm bảo tính tiêng tư

Từ những thập niên 1980, có rất nhiều tổ chức hay quốc gia tiến hành xây dựng các nguyên lý cho việc đảm bảo tính riêng tư. Các nước Mỹ, Canada, Australia, Ấn Độ hay liên minh Châu Âu cũng đều đưa ra các nguyên lý chung. Điển hình là tổ chức hợp tác và phát triển kinh tế Châu Âu đã đưa ra 7 nguyên lý sau được cho là khá đầy đủ:

- Thông báo: Dữ liệu phải được thông báo khi dữ liệu người dùng được thu thập.
- Mục đích: Dữ liệu chỉ được dùng với mục đích đã quy định mà không được dùng với bất cứ mục đích khác.
- Chấp nhận: Dữ liệu không được công bố nếu không được sự đồng ý của người dùng.
- Bảo mật: Dữ liệu thu thập được phải được an toàn trước mọi sự truy cập trái phép.
- Tiết lộ: Dữ liệu phải được thông báo với người đã thu thập dữ liệu đó.
- Truy cập: Dữ liệu phải được phép truy xuất và đảm bảo tính đúng đắn.
- Trách nhiệm: Dữ liệu phải có một phương pháp sẵn có để những người thu thập dữ liệu phải có trách nhiệm tuân theo các nguyên tắc.

---

[1] DART: Dynamic Advertising Reporting & Targeting
[2] Webite "Google ad and content network privacy policy", địa chỉ tại http://www.google.com/privacy_ads.html

1.6. Bảo mật riêng tư cho trình duyệt web

Trình duyệt web là một ứng dụng phần mềm dùng để nhận, trình bày và duyệt các nguồn thông tin trên Internet. Một tài nguyên thông tin được xác định bởi định danh tài nguyên duy nhất (URI[1]) và có thể là một trang web, hình ảnh, video hay một mẩu dữ liệu. Một trình duyệt web có thể định nghĩa như một ứng dụng phần mềm hay chương trình thiết kế để người dùng có thể truy cập, nhận xem các tài liệu và các tài nguyên khác trên Internet.

1.6.1. Bảo mật trình duyệt di động

Ngày nay chúng ta ngày càng gắn bó cuộc sống với các TBDĐ nhiều hơn, chúng là các smartphone hay máy tính bảng. Chúng thật tốt và tiện lợi khi giá cả không còn là vấn đề. Các nền tảng di động đang bị tấn công với tần suất cao và các cuộc tấn công thường thông qua các ứng dụng không tin cậy. Hầu hết các vấn đề xảy ra thông qua sự bùng nổ mạnh mẽ của trình duyệt web. Nó xảy ra với tất cả các TBDĐ, nhưng đặt biệt với các thiết bị chạy HĐH Android. Điều đó đặc biệt đúng khi nói về việc xác định xem một trang web có an toàn hay không. Đã có hơn 20% các vấn đề liên quan đến mua sắm trực tuyến trên các TBDĐ và con số này vẫn đang ra tăng nhanh chóng. Trình duyệt web có thể bị vi phạm bởi một trong các cách sau:

- HĐH bị vi phạm và chương trình độc hại sẽ đọc/sửa không gian nhớ của trình duyệt với chế độ đặc quyền.
- HĐH đang có chương trình độc hại chạy ngầm và nó cũng đọc/sửa không gian nhớ của trình duyệt với chế độ đặc quyền.
- Trình duyệt chính có thể bị thực thi bởi sự tấn công.
- Các thành phần của trình duyệt có thể bị tấn công.
- Các thành phần đính kèm có thể bị tất công.

1.6.2. Truy cập lịch sử trình duyệt

Bất cứ khi nào truy cập Internet từ máy tính, TBDĐ thì trình duyệt web sẽ lưu những trang chúng ta đã ghe thăm. Các trình duyệt phổ biến hiện nay như Chrome, Firefox, Internet Explorer đều lưu nơi bạn vào mạng cũng như lịch sử những trang đã truy cập. Đây không phải là việc gì mờ ám cần che dấu hoặc xâm phạm quyền riêng tư. Nó phục vụ cho người dùng được thuận tiện hơn, trừ khi người dùng đang làm một cái gì đó mà không muốn người khác nhìn thấy. Ứng dụng trình duyệt được thiết kế để hỗ trợ người dùng biết nơi nào mình đã ghe thăm các trang web và những gì mình đã đọc hay xem trực tuyến trên Internet. Và qua thời gian, các tính năng hữu ích đã được thêm vào tính năng lịch sử.

---

[1] URI: Uniform Resource Identifier

Tính năng lịch sử trên các trình duyệt để làm đơn giản hóa trải nghiệm trực tuyến và cung cấp sự tiện lợi cho người dùng. Nhưng ai đó cũng có thể xem lịch sử trình duyệt, mà hầu hết chúng ta không muốn như vậy. Tuy nhiên, điều đó không thường xảy ra. Nếu ta quan tâm đến sự riêng tư, bất kể thứ gì thì có thể sử dụng lựa chọn xóa lịch sử trình duyệt hay duyệt web riêng tư. Tùy theo mỗi trình duyệt mà việc duyệt web riêng tư được hỗ trợ và bảo mật.

Việc lập trình truy cập lịch sử trình duyệt của thiết bị có thể dùng các thư viện hàm sẵn có với ngôn ngữ lập trình tương ứng. Ví dụ dưới đây là một đoạn mã nguyền đọc lịch sử trình duyệt, dùng lớp BookmarkColumns trong gói android.provider để lấy tất cả các địa chỉ, tiêu đề của các đánh dấu (bookmark).

```
public ArrayList<HistoryEntry> getBrowserHistory() {

  String title = "";
  String url = "";

  ArrayList<HistoryEntry> list = new ArrayList<HistoryEntry>();

  String[] proj = new String[] { Browser.BookmarkColumns.TITLE,
          Browser.BookmarkColumns.URL };
  String sel = Browser.BookmarkColumns.BOOKMARK + " = 0"; // 0 = history,
                                                          // 1 = bookmark
  Cursor mCur = getContentResolver().query(Browser.BOOKMARKS_URI, proj,
          sel, null, null);
  mCur.moveToFirst();

  if (mCur.moveToFirst() && mCur.getCount() > 0) {
      boolean cont = true;
      while (mCur.isAfterLast() == false && cont) {
          HistoryEntry entry = new HistoryEntry();

          title = mCur.getString(mCur
                  .getColumnIndex(Browser.BookmarkColumns.TITLE));
          url = mCur.getString(mCur
                  .getColumnIndex(Browser.BookmarkColumns.URL));
          // Do something with title and url
          entry.setTitle(title);
                          entry.setUrl(url);
                          list.add(entry );
          Log.d("TAG", "title    " + title);
          mCur.moveToNext();
      }
  }

  mCur.close();

  return list;
}
```

Và thêm một ví dụ, đoạn mã nguồn Java dưới đây lấy cookie của trình duyệt Chrome trên thiết bị chạy HĐH Android. Truy xuất cookie của trình duyệt, lập trình viên sẽ dùng phương thức getCookie() thuộc lớp CookieManager trong gói android.webkit.

```
@Override
public void onPageStarted(WebView view, String url, Bitmap favicon) {
  String cookieString = cookieManager.getCookie(url);

  if (cookieString != null && cookieString.indexOf("_toggl_session") > -1) {
    String s = cookieString.substring(cookieString.indexOf("_toggl_session") + 15);
    int end = s.length();
    if (s.indexOf(";") > -1) {
      end = s.indexOf(";");
    }
    String togglSession = s.substring(0, end);

    if (togglSession.length() < 200) {
      Intent intent = getIntent();
      intent.putExtra(TOGGL_SESSION_ID, togglSession);
      setResult(RESULT_OK, intent);
      view.stopLoading();
      cookieManager.removeAllCookie();
      finish();
    }
  }
  super.onPageStarted(view, url, favicon);
}
```

1.7. Một số hệ thống an ninh thực tế trên thiết bị di động

1.7.1. Samsung Knox

Samsung Knox là một nền tảng an ninh cho TBDĐ phát triển bởi Samsung. Nó cung cấp các tính năng bảo mật cho cả dữ liệu cá nhân cũng như doanh nghiệp trên cùng TBDĐ. Người dùng có thể chọn biểu tượng để chuyển đổi từ không gian cá nhân sang công việc ngay tức thời mà không cần khởi động lại thiết bị. Tính năng trên tương thích hoàn toàn với HĐH Android.

Nó được xây dựng bên trong các TBDĐ mới chạy HĐH Android. Và đã được nhận giải thưởng giải pháp cấp chính phủ về đảm bảo an ninh, quản lý và tùy biến các TBDĐ được thương mại hóa. Nó tạo ra một phân vùng ảo trên thiết bị để quản lý các ứng dụng và dữ liệu khỏi các tấn công trái phép. Nó có thể tạo

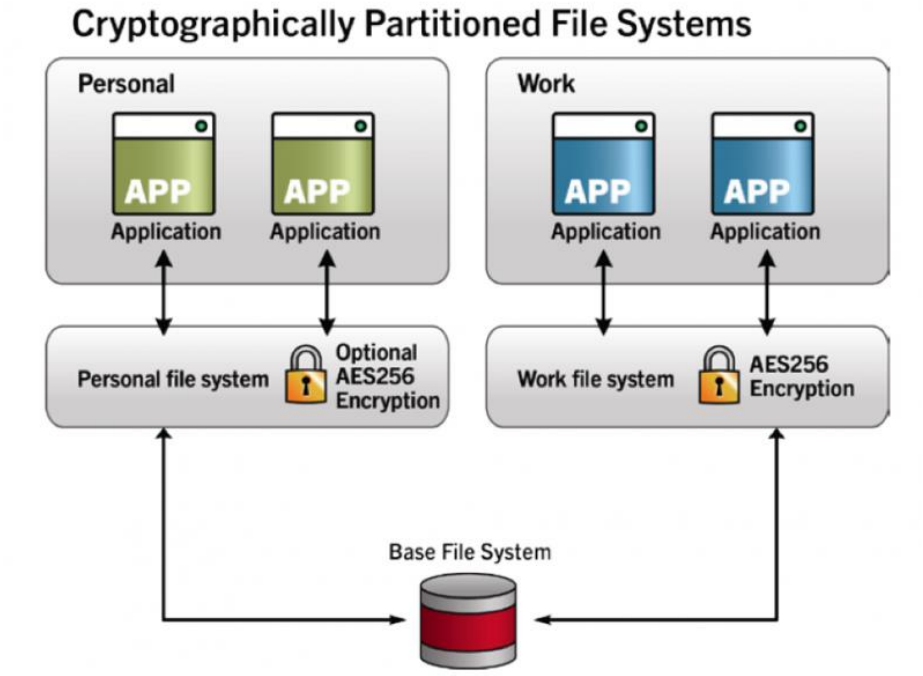
ra các giải pháp phần mềm tùy biến phù hợp với các yêu cầu cho công nghiệp. Việc quản lý di động được dễ dàng hơn với việc hỗ trợ hầu hết các giải pháp thông dụng như MDN, EMM, SSO và VPN. Và dưới đây là giải pháp bảo mật tổng thể:

- Bảo mật đa lớp (Multi-layered security): Knox là một công nghệ đa lớp xây dựng cho cả phần cứng và phần mềm cho các thiết mới mới của Samsung. Knox liên tục kiểm tra tính toàn vẹn của thiết bị và phát hiện bất kỳ sự giả mạo nào, đảm bảo dữ liệu được an toàn hơn.
- Cái tiến bảo mật cho Android (Security Enhancements for Android): Cải tiến bảo mật cho Android bảo vệ các ứng dụng và dữ liệu bằng cách xác định rõ từng quá trình được phép làm gì và những dữ liệu nào có thể truy cập. Nó giúp bảo vệ thiết bị bằng cách sử dụng tên miền, quyền, chính sách bảo mật và kiểm soát truy cập bắt buộc.
- Vùng an toàn (TrustZone): Knox đưa ra một kiến trúc bộ xử lý gọi là TrustZone, có hai thế giới: Thế giới bình thường và thế giới an toàn. Hầu như tất cả các phần mềm chạy trên smartphone chúng ta biết ngày nay vẫn chạy trong Thế giới bình thường. Thế giới an toàn được dành riêng cho tính toán nhạy cảm mức độ cao và được sử dụng rộng rãi bởi Knox để bảo vệ dữ liệu bí mật của doanh nghiệp.
- Khởi động an toàn (Secure Boot & Trusted Boot): Khởi động an toàn ngăn các bộ tải và các nhân không tin cậy nạp vào thiết bị. Điều này có nghĩa là thiết bị không bị giả mạo và có thể tải được bộ chứa Knox. Nó đảm bảo trình nạp và nhân của HĐH là bản gốc từ nhà máy. Điều này được thực hiện bằng cách ghi lại các phép đo thiết bị ban đầu và kiểm tra liên tục thiết bị khi khởi động để đảm bảo các phép đo này không thay đổi.
- Bảo mật gốc phần cứng (Hardware Root of Trust): Là một bộ cơ chế bảo mật được tích hợp trong phần cứng thiết bị, bất kể khi nào các điều khiển mặc định của thiết bị bị thay đổi. Bao gồm khóa bảo mật an toàn và khóa gốc thiết bị, thực hiện các hoạt động xác thực và mã hóa liên kết với thiết bị. Để sử dụng Knox, thiết bị phải được hỗ trợ công nghệ ảo hóa ở mức phần cứng và giới hạn ở các thiết bị của Samsung.

1.7.2. BlackBerry Balance

BlackBerry Blance là một tính năng sẵn có của BlackBerry 10 trở lên. Nó cho phép quản lý dữ liệu cá nhân hay chuyên biệt một cách độc lập. E-mail, ứng dụng và các thông tin kinh doanh nhạy cảm có thể được sao chép trên vùng cá nhân. Quản trị viên có thể xóa toàn bộ dữ liệu kinh doanh từ xa khi thiết bị bị

mất hay đánh cắp. Người dùng có thể dễ ràng chuyển giữa chế động công việc và các nhân. Các ứng dụng dành cho công việc được phát triển bởi một hệ thống tùy biến doanh nghiệp của BlackBerry (Enterprise-Customized BlackBerry World). Nó cho phép người dùng chuyển toàn bộ dữ liệu cá nhân trong thiết bị. Công nghệ này giúp cân bằng các chính sách của dữ liệu cá nhân và an ninh thông tin kinh doanh.



Hình 1.2. Hệ thống phân vùng mã hóa BlackBerry Balance

Các thiết bị BlackBerry 10 tạo ra khóa bảo mật ngẫu nhiên để mã hóa nội dung của tệp. Các khóa bảo mật được bảo vệ bởi hệ thống phân phối các khóa bảo mật. Hệ thống hiện nay vẫn còn hạn chế chỉ hỗ trợ cho các thiết bị chạy HĐH Blackberry và chỉ từ phiên bản 10 trở lên. Từ hạn chế trên dẫn đến số lượng người sử dụng không nhiều nên việc phổ biến cũng như nghiên cứu phát triển cũng bị giới hạn.

1.7.3. SCANDAL

SCANDAL là một bộ phân tích tĩnh có thể phát hiện rò rỉ thông tin cá nhân trong các ứng dụng Android. Nó kiểm tra bất cứ luồng dữ liệu từ nguồn thông tin, có thể kiểm soát tất cả các tình trạng có thế xảy ra khi sử dụng ứng dụng [5, tr.1-3].

Hạn chế là trong quá trình xử lý nó có thể mất nhiều thời gian cũng như không gian nhớ của thiết bị. SCANDAL không hỗ trợ các phương thức của Java gốc (JNI[1]) [5, tr.8-9]. Các lập trình viên ứng dụng có thể dùng JNI để cùng làm

[1] JNI: Java Native Interface

việc với các thư viện C/C++ trong ứng dụng. Vì ngôn ngữ gốc là của bộ xử lý mã nguồn Dalvik VM, nên không thể chạy các thư viện JNI trong xử lý của SCANDAL.

1.7.4. IccTA

IccTA là một công cụ mã nguồn mở phát hiện rò rỉ thông tin. Dù cho các ứng dụng Android được lập trình bằng Java, nhưng chúng được biên dịch bằng mã lệnh Dalvik thay vì các mã Java truyền thống. IccTA dùng Dexpler để chuyển mã Dalvik ra Jimple [6, tr.5-6] rồi sửa chúng và kết nối trực tiếp đến các thành phần để xử lý luồng dữ liệu giữa các thành phần đó.

Hạn chế hiện tại là IccTA chỉ xử lý hiệu quả được các lời gọi với các tham số là chuỗi cố định, nó cũng không xử lý được đa luồng hay các chuỗi phức tạp (ví dụ dùng StringBuilder) [6, tr.9-10].

1.7.5. AndroidLeaks

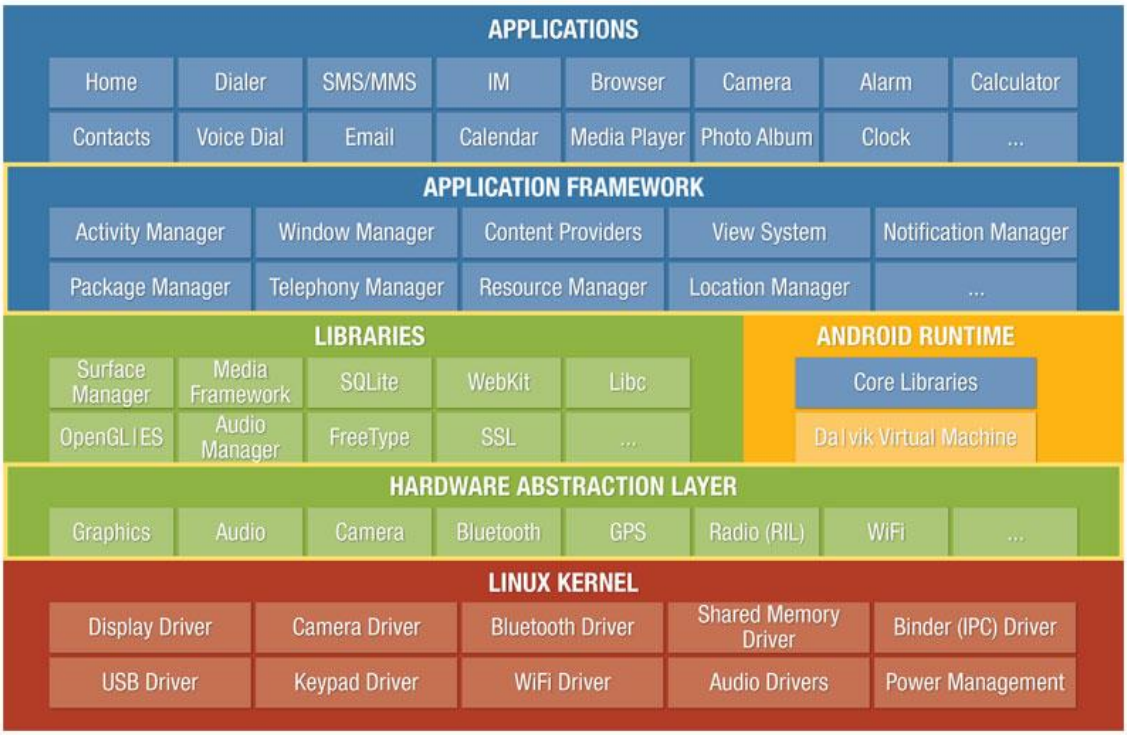
AndroidLeaks là công cụ tự động phát hiện việc vi phạm bảo mật thông tin trong các ứng dụng Android. Trước khi tìm các vi phạm bảo mật, nó thực hiện một vài bước sau: Trước tiên, chuyển đổi các mã của ứng dụng Android (APK) từ định dạng DEX sang JAR với việc dùng ded hoặc dex2jar [1, tr.7-8]. AndroidLeaks dùng WALA và xây dựng một biểu đồ lời gọi của mã ứng dụng và các thư viện bên trong. Nó lặp lại thông qua các lớp của ứng dụng và xác định các phương thức gọi tài nguyên và kiểm tra các phương thức khác có thể yêu cầu các quyền. Nếu ứng dụng chứa kết hợp các quyền có thể lấy dữ liệu cá nhân như đọc trạng thái thiết bị và truy cập Internet, nó sẽ thực hiện xử lý luồng dữ liệu để xác định xem thông tin cá nhân có bị lấy không.

Hạn chế xuất phát từ những giới hạn sẵn có trong việc xử lý tính toán thường phát sinh vấn đề giữa tốc độ, độ chính xác và sai số [1, tr.14-15]. Trong đó hướng tiếp cận xử lý động có thể có độ chính xác cao hơn vì trên thực tế các thông tin bị rò rỉ tại thời gian chạy. Việc kết hợp AndroidLeaks với một hướng tiếp cận xử lý động sẽ có triển vọng vì AndroidLeaks có thể xử lý nhanh với số lượng ứng dùng nhiều và sau đó cung cấp thêm các ứng dụng tiềm năng cho xử lý động. Nó cũng không hỗ trợ xử lý luồng điều khiển và luồng dữ liệu.

Chương 2. Hệ thống TaintDroid

2.1. Giới thiệu Android

Android là một nền tảng di động mã nguồn mở với nhân HĐH Linux, dành cho các TBDĐ như smartphone và máy tính bảng. Hầu hết các chức năng được phát triển như các ứng dụng chạy bên trên tầng trung gian. Các ứng dụng được viết bằng ngôn ngữ Java hoặc C/C++, chúng được dịch thành các mã tùy biến như định dạng DEX[1]. Mỗi ứng dụng thực thi các thực thể và biên dịch bên trong máy ảo Dalik và dưới đây là miêu tả tổng quát kiến trúc của một HĐH Android.



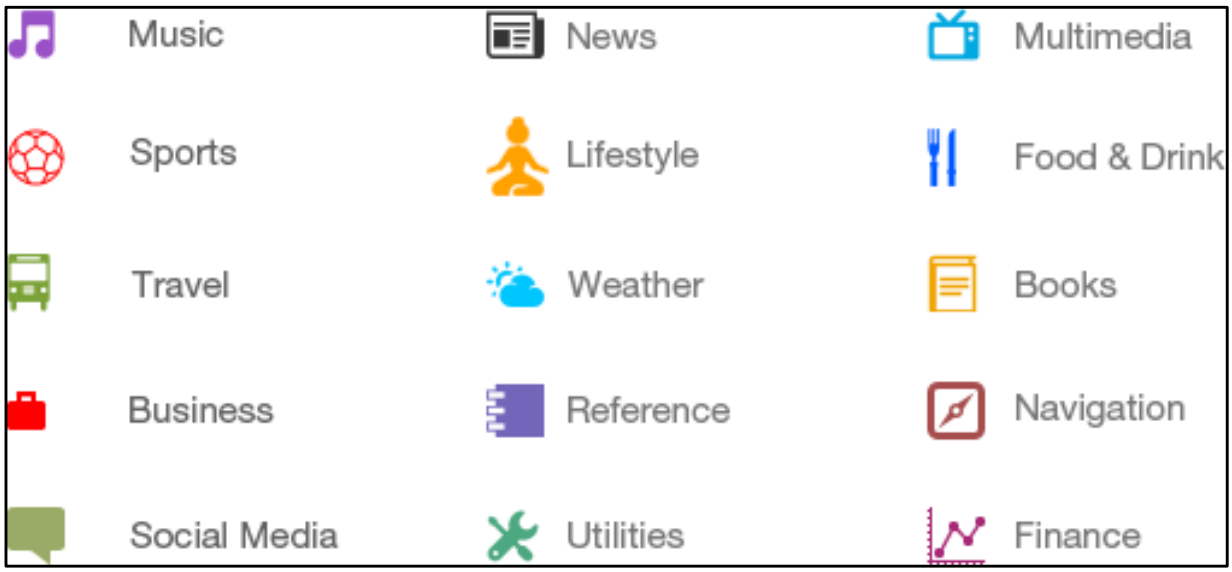
Hình 2.1. Kiến trúc HĐH Android

- Linux Kernel **(**Nhân Linux): Nhân Linux được phát triển từ HĐH mã nguồn mở GNU, do vậy chúng ta có thể sử dụng trên điện thoại của mình như phần mềm miễn phí và nó đã có sẵn các trình điều khiển cho phần cứng. Nhân Linux là phần quản lý việc điều khiển chương trình, người dùng, tài nguyên, yêu cầu vào/ra, xử lý bên trong máy tính.
- Hardware Abstraction **(**Tầng phần cứng trừu tượng): Là một bộ các định tuyến trong phần mềm giả lập một số chi tiết nền tảng, cumg cấp việc truy nhập trực tiếp cho các ứng dụng tới các tài nguyên phần cứng.
- Libraries **(**Tầng thư viện): Là một thư viện hỗ trợ cho việc lập trình hiệu

[1] DEX - Dalvik EXecutable byte-code format: Định dạng mã thự thi Dalvik

quả hơn. Nói gồm các thư viện như: thư viện đồ họa, các thư viện đa phương tiện (âm thanh, hình ảnh và video), trình duyệt, kết nối bảo mật, font chữ, cơ sở dữ liệu và các thư viện khác. Môi trường chạy Android làm cho việc sử dụng máy ảo như môi trường chạy các tập APK của ứng dụng Android. Ưu điểm của việc dùng máy ảo là mã ứng dụng được tách biệt với HĐH lõi, đảm bảo lỗi xảy ra không ảnh hưởng đến HĐH chính.

- Application Framework (Tầng nền ứng dụng): Hỗ trợ các tính năng cho việc sử dụng và giải trí của các ứng dụng với nhiều nhu cầu sử dụng khác nhau và dưới đây là một vài tính năng chính:
  - Trình duyệt web giữa trên nền Webkit
  - Xứ lý đồ họa được hỗ trợ bởi các thư viện đồ họa tiên tiến
  - Lưu trữ cơ sở dữ liệu bằng SQL
  - Hỗ trợ rất nhiều loại âm thanh, video và ảnh
  - Thiết bị giả lập, các công cụ để gỡ rối (debug), …
- Application (Tầng ứng dụng): Là tầng cao nhất, hầu hết các người dùng thiết bị sẽ giao tiếp với tầng này (cho các chức năng chính: gọi điện, duyệt web, vv). Dưới đây là danh mục các ứng dụng đang đã được phát triển trên nền tảng Android tính tới thời điểm hiện tại.



Hình 2.2. Danh mục ứng dụng Android

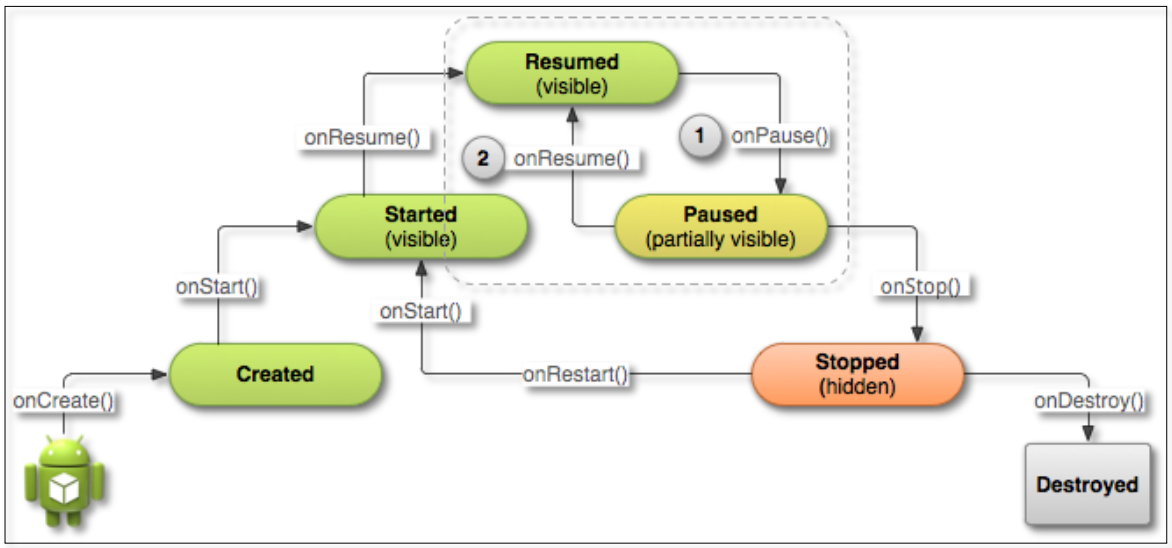
Các thành phần của chương trình: Trước khi phát triển hay cải tiến một ứng dụng Android, điều thiết yếu là chúng ta cần phải hiểu về các thành phần chính của ứng dụng. Các thành phần của ứng dụng là các khối xây dựng cho ứng dụng, mỗi thành phần là một điểm riêng biệt mà hệ thống có thể truy xuất ứng dụng.

Hình 2.3. Các Thành Phần Chương Trình Android

- Activities (Hoạt động): Là thành phần thực thi một màn hình đơn với một giao diện người dùng. Ví dụ một ứng dụng thư điện thử (e-mail) có thể có một hành động hiển thị một danh sách các email mới, các hoạt động khác như soạn hay đọc e-mail. Dù cho với người dùng thì các hoạt động trên làm việc cùng nhau trong mô hình sử dụng một ứng dụng e-mail thông thường. Nhưng một ứng dụng có thể khởi tạo bất cứ hoạt động nào trong các hoạt động trên. Ví dụ một ứng dụng chụp ảnh có thể khởi tạo hoạt động soạn e-mail với việc đính kèm hình ảnh vừa chụp.
- Services (Dịch vụ): Một dịch vụ là một thành phân chạy phía dưới của ứng dụng và chạy với thời gian dài và không có giao diện người dùng. Ví dụ một dịch vụ có thể chơi nhạc trong khi người dùng đang sử dụng một ứng dụng khác.
  - Content Providers (Phân phối nội dung): Một thành phần phân phối nội dung quản lý một bộ chia sẻ dữ liệu của ứng dụng. Bạn có thể lưu trữ dữ liệu trong hệ thống tệp, một CSDL SQLLite, trên website hay bất cứ vùng lưu trữ nào mà ứng dụng có thể truy cập. Các ứng dụng có thể truy cập, sửa đổi dữ liệu nếu nó cho phép.
- Broadcast Receivers (Bộ tiếp nhận): Một bộ tiếp nhận là một thành phần xử lý các thông báo được truyền ở mức hệ thống. Có rất nhiều thông báo sẵn của hệ thống như thông báo tắt màn hình, pin yếu hoặc một ảnh được chụp. Các ứng dụng cũng có thể khởi tạo các broadcast của riêng mình như báo cho các ứng

dụng khác khi dữ liệu đã tải xong và sẵn sàng để chạy hay cài đặt. Tuy thành phần này không hiện nên giao diện người nhưng nó có thể tạo ra các thông báo trên thanh trạng thái để báo cho người dùng khi một broadcast xảy ra.

- Intent: Là một thành phần miêu tả trừu tượng của một hoạt động đã được thực thi. Nó có thể dùng để khởi tạo hoạt động và một chương trình sẽ được hoạt động theo một chu trình như miêu tả ở hình 2.4 dưới đây.



Hình 2.4. Chu Trình Hoạt Động Chương Trình Android

- Started Activity: Không như các mô hình lập trình với việc ứng dụng được chạy với một phương thức main(), hệ thống Android khởi tạo mã trong một thực thể hoạt động bởi việc gọi các phương thức triệu hồi (callback) xác định để đồng thời xử lý các kịch bản xác định của chu kỳ.
- Paused Activity: Các ứng dụng thường thỉnh thoảng cần dừng hoạt động ví dụ khi các ứng dụng chạy ở chế độ nhiều cửa sổ, chỉ một trong số các ứng dụng ở trạng thái kích hoạt trong một thời điểm, hệ thống sẽ dừng các ứng dụng khác. Khi ứng dụng về trạng thái dừng, hệ thống gọi phương thức onPause(), nó cho phép dừng các hành động xử lý với bên ngoài trong lúc dừng hoặc cấm mọi thông tin được lưu trong khi người dùng chưa tiếp tục chạy.
- Resumed Activity: Khi người dùng khôi phục hoạt động từ trạng thái dừng, hệ thống sẽ gọi phương thức onResume(). Hệ thống sẽ gọi phương thức này mỗi khi hoạt động được thực hiện trở lại, kể cả khi nó được khởi tạo lần đầu.
- Stopped Activity: Khi hoạt động nhận được một lời gọi phương thức onStop(), nó không được hiện thị trong thời gian lâu và phải được giải phóng

các tài nguyên không được dùng khi người dùng không sử dụng ứng dụng. Khi hành động được kết thúc, hệ thống có thể sẽ xóa thực thể thực thi chương trình nếu cần để giải phóng bộ nhớ. Trong trường hợp nghiêm trọng hệ thống sẽ dừng hoàn toàn tiến trình của ứng dụng mà không gọi phương thức onDestroy().

- Restared Activity: Khi hoạt động được gọi trở lại từ trạng thái kết thúc, nó nhận được một lời gọi onRestart(). Hệ thống cũng gọi phương thức onStart() mỗi khi hoạt động được kích hoạt. Phương thức onRestart() cũng được gọi khi hoạt động được khôi phục từ trạng thái kết thúc.

2.2. Giới thiệu TaintDroid

TaintDroid là một hiện thực hóa trong việc theo dõi các dấu vết bên trong HĐH Android. TaintDroid thực hiện việc theo dõi ở mức biến hệ thống bên trong máy ảo biên dịch. Đánh dấu vết được lưu trữ trong một thẻ taint (taint tag). Khi các ứng dụng thực hiện các phương thức nguyên gốc, các thẻ taint được đưa trở lại. Cuối cùng các thẻ taint được gán cho gói và được tiếp nhận thông qua binder. Hình 2.5 miêu tả kiến trúc của TaintDroid và thông tin được đánh dấu trong một ứng dụng tin cậy với đủ thông tin. Giao diện đánh dấu gọi một phương thức nguyên gốc thực thi trình biên dịch máy ảo Dalvik. Máy ảo Dalvik phân tán các thẻ taint tương ứng với các quy định về luồng dữ liệu như các ứng dụng tin cậy sử dụng thông tin được đánh dấu. Các thực thể từ xa của máy ảo Dalvik sẽ đưa các thẻ taint tới các ứng dụng không tin cậy [12, tr. 4]. Khi các ứng dụng đó gọi một thư viện xác định như gửi tín hiệu mạng, thư viện nhận được taint cho dữ liệu yêu cầu và thực thi sự kiện.

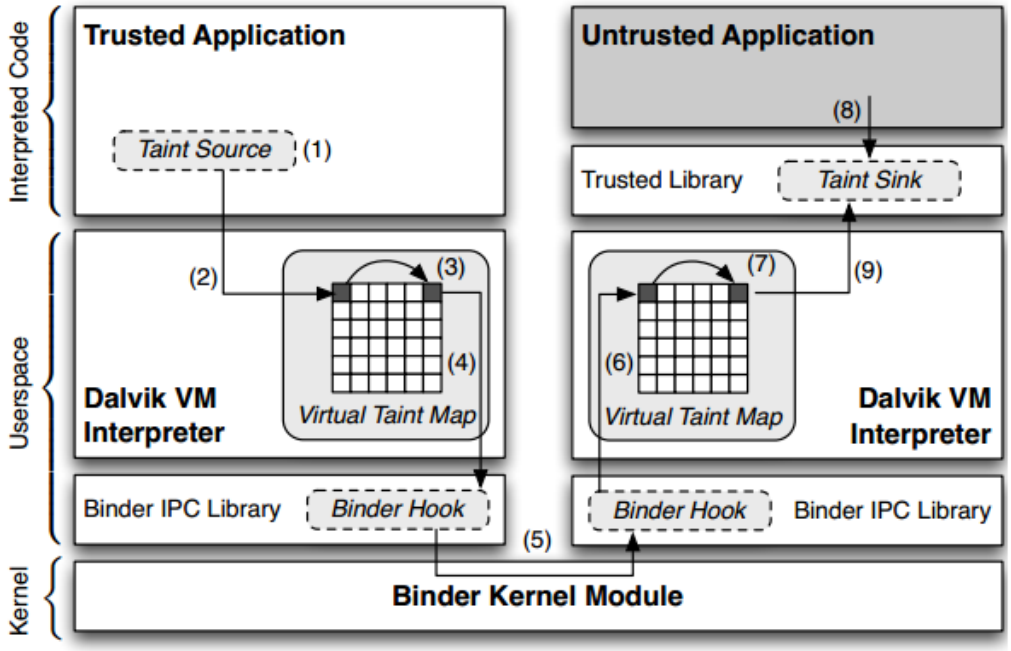
Trong khi một vài HĐH điện thoại di động cho phép người dùng điều khiển việc ứng dụng truy cập thông tin nhạy cảm như cảm biến vị trí, ảnh chụp và danh sách liên lạc, người dùng không có cách nào theo dõi một cách trực quan xem các ứng dụng dùng dữ liệu cá nhân của họ như thế nào. Để phục vụ việc này, TaintDroid là một công cụ theo dõi luồng thông tin ở mức hệ thống. Nó có thể theo dõi đồng thời nhiều nguồn tài nguyên thuộc dạng dữ liệu nhạy cảm. Điểm nổi bật của nó là sự hiệu quả và đã gặt hái được bằng việc tích hợp yếu tố của việc phân tán taint ở mức biến, mức thông điệp, mức phương thức và mức tệp tin để cải tiến được 14% hiệu năng theo chỉ số đánh giá CPU-bound.

TaintDroid đã được sử dụng để thực nghiệm các hành vi của 30 ứng dụng bên thứ 3 tiêu biểu, chọn ngẫu nhiên từ kho ứng dụng Android. Kết quả là 2 ứng dụng điều khiển dữ liệu nhạy cảm đáng khả nghi và 15 trong số 30 ứng dụng đã gửi thông tin vị trí của người dùng đến các máy chủ quảng cáo từ xa

[12, tr. 8]. Điều đó chứng minh tính hiệu quả và giá trị của các công cụ theo dõi nâng cao trên các nền tảng di động như TaintDroid.

2.2.1. Kiến trúc TaintDroid

TaintDroid phiên bản mới nhất được phát triển trên nền HĐH Android 4.3, nên nhìn chung toàn bộ kiến trúc hệ thống tương tự như HĐH Android 4.3.



Hình 2.5. Kiến Trúc TaintDroid Bên Trong Android

- Tài nguyên taint (Taint Source): Là tất cả các thông tin hay dữ liệu được TaintDroid đánh dấu, mọi thông tin được đánh dấu đều có thể được ánh xạ qua bảng anh xạ taint ảo do nó định nghĩa.
- Bảng ánh xạ taint ảo (Virtual Taint Map): Là bảng được TaintDroid định nghĩa về các loại taint sẽ được xử lý, mỗi taint đều được gắn với một chỉ số xác định và dưới đây là một số taint trong bảng ánh xạ:
  - <0x00000001, "Location">
  - <0x00000002, "Address Book (ContactsProvider)">
  - <0x00000008, "Phone Number">
  - <0x00000400, "IMEI">
  - <0x00004000, "User account information">
  - <0x00008000, "Browser history">
  - …

- Can thiệp bộ kết nối (Binder Hook): TaintDroid can thiệp thay thế các xử lý taint nhằm kiểm soát chính xác và nhanh chóng theo luồng dữ liệu trong hệ thống Android. Từ đó có thể phân tích và xử lý dữ liệu taint nhanh chóng và đảm bảo sự ổn định về luồng và hiệu năng của hệ thống hiện tại.

▪ Gói taint (Taint Sink): Các gói taint được định nghĩa bên trong môi trường ảo, kết nối các thẻ taint với hệ thống giám sát.

2.2.2. Các chức năng

Thông báo khi dữ liệu nhạy cảm bị gửi đi: TaintDroid liên tục kiểm tra luồng dữ liệu được gửi đi từ các ứng dụng. Khi dữ liệu nhạy cảm được gửi đi dựa trên việc ánh xạ với bảng taint ảo đã định nghĩa sẵn. Hệ thống sẽ thông báo chi tiết đến người dùng về loại dữ liệu bị gửi đi kèm theo các thông tin như: ứng dụng gửi, thời gian, IP sẽ gửi dữ liệu đến.

Lưu dấu vết dữ liệu để xác thực quyền truy cập: Khi hệ thống phát hiện taint được gửi đi, dấu vệt của nó sẽ được lưu lại và xác thực quyệt truy cập thông tin đó. Việc xác định giữa trên quyền của ứng dụng được khai báo từ lúc cài đặt vào HĐH trên thiết bị.

Hiển thị nơi gửi dữ liệu đến: Các thông tin nhạy cảm được gửi đi từ ứng dụng không tin cậy đều được xác định kèm theo địa chỉ IP của nơi nhận. Địa chỉ này thường là các máy chủ mà ứng dụng kết nối đến, đôi khi là máy chủ quảng cáo.

2.2.3. Nguyên lý hoạt động

Thông tin được đánh dấu bên trong ứng dụng đã được xác định. Giao diện taint triệu gọi phương thức nguyên gốc từ giao diện trình biên dịch máy ảo Dalvik, lưu trữ các dấu taint bên trong bản đồ taint ảo. Máy ảo Dalvik đưa các thẻ taint tương ứng qua luồng dữ liệu như dành cho các ứng dụng tin cậy sử dụng thông tin taint. Mọi thực thể trình biên dịch sẽ phân tán các thẻ taint dưới dạng mô phỏng. Khi ứng dụng tin cậy sử dụng thông tin taint bên trong giao dịch IPC [12, tr.3-4], các thư viện kết nối đảm bảo gói chứa thẻ taint ánh xạ các taint liên kết với tất cả dữ liệu. Gói sẽ được truyền qua nhân và nhận được bởi ứng dụng không tin cậy từ xa, lưu ý mã biên dịch này là không an toàn. Thư viện kết nối nhận thẻ taint từ gói và gán nó với tất cả các giá trị tương ứng thông qua bảng ánh xạ taint. Thực thể từ xa của máy ảo Dalvik truyền các thẻ taint phân biệt cho ứng dụng không tin cậy. Khi ứng dụng không tin cậy triệu gọi thư viện, thư viện nhận thẻ taint như yêu cầu bình thường và đưa ra sự kiện.

Triển khai kiến trúc trên còn có một số thách thức mức hệ thống, bao gồm 1) Lưu trữ các thẻ Taint, 2) Phân tán các mã biên dịch taint, 3) Phân tán các taint mã nguyên gốc, 4) Phân tán taint IPC, 5) Phân tán các taint thiết bị lưu chữ thứ cấp.

2.2.4. Chính sách can thiệp thay thế

Dùng TaintDroid cho việc xử lý các chính sách yêu cầu xác định được các tài nguyên nhạy cảm và nghiên cứu các nguồn taint bên trong HĐH. Các hệ thống xử lý taint động đảm bảo tài nguyên taint và việc thay thế ẩn không nghiêm trọng. Tuy vậy những hệ thống phức tạp như Android cung cấp thông tin cho các ứng dụng thông qua rất nhiều cách như truy cập trực tiếp và giao diện dịch vụ. Mỗi kiểu của thông tin nhạy cảm phải được xem xét cẩn thận để xác định phương pháp tốt nhất trong việc định nghĩa tài nguyên taint. Các nguồn tài nguyên taint có thể chỉ thêm các thẻ taint vào bộ nhớ mà TaintDroid cung cấp lưu trữ thẻ. Hiện nay tài nguyên taint và thay thế ẩn bị giới hạn với các biến trong mã biên dịch, các thông điệp IPC và các tệp [12, tr.7]. Phần này sẽ nói về việc các tài nguyên taint sẽ được thực thi với các giới hạn trên. Chúng ta sẽ khái quát hóa các tài nguyên taint dựa trên các đặc tính của thông tin.

- Các cảm biến băng thông hẹp (Low-bandwidth Sensors): Rất nhiều kiểu thông tin nhạy cảm được lấy được thông qua các cảm biến băng thông hẹp, như vị trí và tọa độ. Thông tin thường thay đổi liên tục và sử dụng liên tục bởi nhiều ứng dụng, tuy nhiên HĐH smartphone đa truy cập đến các cảm biến băng thông hẹp dùng một bộ quản lý. Bộ quản lý cảm biến này đưa ra một điểm lý tưởng cho thay thế can thiệp tài nguyên taint. Trong TaintDroid chúng ta đã thay thế can thiệp bằng ứng dụng LocationManager và SensorManager.
- Các cảm biến băng thông rộng (High-bandwidth Sensor): Các tài nguyên thông tin nhạy cảm như microphone và camera là thông tin băng thông rộng. Mỗi yêu cầu gửi từ cảm biến thường trả lại lượng dữ liệu lớn chỉ dùng bởi một ứng dụng. Tuy nhiên HĐH smartphone có thể chia sẻ thông tin cảm biến thông qua các bộ nhớ đệm, tệp lớn, tệp phải được đánh dấu bằng thẻ tương ứng. Để tạo sự linh hoạt cho các API, chúng ta can thiệp thay thế cả bộ nhớ đệm dữ liệu và đánh dấu tệp cho việc theo dõi thông tin của microphone và camera.
- Các cơ sở dữ liệu thông tin (Information Databases): Thông tin chia sẻ như địa chỉ liên lạc và tin nhắn SMS thường được lưu trong cơ sở dữ liệu (CSDL) dưới dạng tệp. Cách tổ chức này cung cấp một tài nguyên taint trừu tượng tương tự với các cảm biến phần cứng. Bằng việc thêm một thẻ taint vào các tệp CSDL thì tất cả các thông tin đọc từ tệp sẽ tự động được đánh dấu, chúng ta dùng kỹ thuật này để đánh dấu thông tin sổ địa chỉ.
- Các định danh thiết bị (Device Identifiers): Thông tin duy nhất xác định điện thoại hay người dùng là rất nhạy cảm. Không phải tất cả thông tin cá nhân có thể dễ dàng đánh dấu, tuy nhiên điện thoại có một vài định danh rất dễ đánh

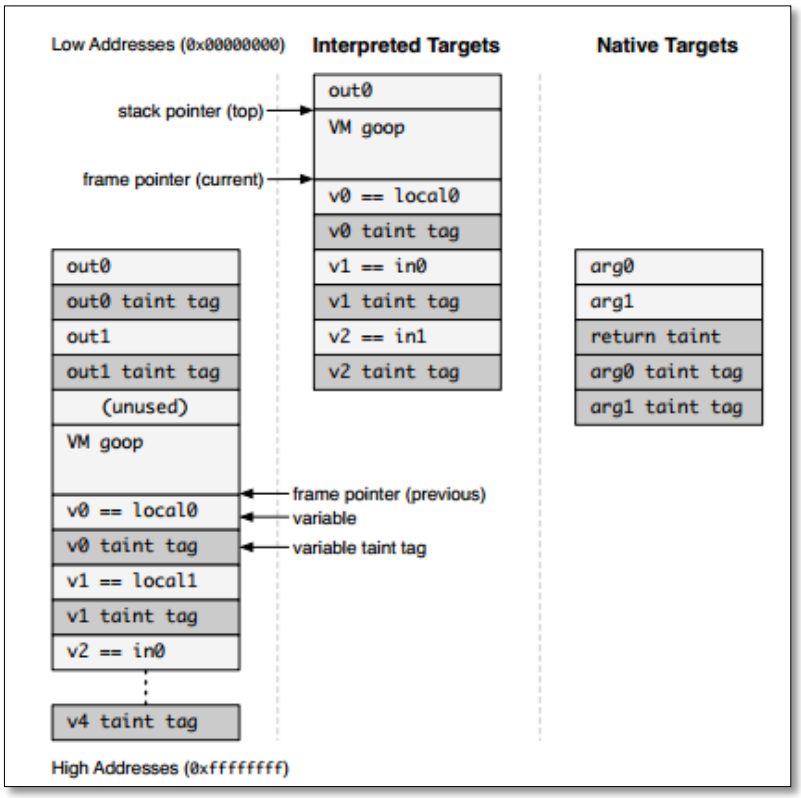
dấu như: số điện thoại, định danh thẻ SIM và định danh thiết bị được truy cập thông qua các thư viện hàm thông dụng [12, tr.8].

2.3. Các thành phần chính

2.3.1. Lưu trữ các thẻ Taint

Việc lựa chọn cách lưu trữ các thẻ dấu vết sẽ ảnh hưởng đến hiệu năng và bộ nhớ của hệ thống. Các hệ thống kiểm tra dấu vết động thường lưu trữ các thẻ mức từng byte hay word dữ liệu. Các thẻ taint thường được lưu trữ trong bộ nhớ bóng không liền kề và bản đồ dấu vết. TaintDroid dùng biến trong trình biên dịch Dalvik [12, tr.4-5] để lưu trữ các thẻ taint vào các biến bên trong bộ nhớ.

Dalvik có 5 kiểu biến cần cho lưu trữ dấu vết gồm biến nội bộ phương thức, đối phương thức, trường của lớp tĩnh, trường thực thể lớp và mảng. Mỗi biến đều được lưu trữ dạng vector với độ dài 32bit mã hóa nên cho phép tạo 32 taint khác nhau.



Hình 2.6. Định dạng thay đổi trong Stack

Thẻ taint xen giữa đăng ký cho các đối tượng của phương thức biên dịch và chèn vào các phương thức nguyên gốc. Các thẻ taint được lưu trữ liền với các trường của lớp và các mảng bên trong các cấu trúc dữ liệu của máy ảo biên dịch. Nó chỉ lưu một thẻ taint ở mỗi mảng để tối ưu việc lưu trữ. Mỗi giá trị lưu trữ của thẻ taint dùng khắc phục sự không hiệu quả cho các đối tượng kiểu chuỗi của Java, tất cả các ký tự đều có cùng một thẻ.

2.3.2. Phân tán các mã biên dịch taint

TaintDroid được kiểm tra theo luồng nên chính xác và không làm ảnh hưởng đến hiệu năng. Nó thực hiện kiểm tra các taint ở mức biến với máy ảo biên dịch Dalvik. Các biến cung cấp ngữ nghĩa có giá trị cho việc phân tán taint, các con trỏ dữ liệu riêng biệt từ các biến vô hướng và chủ yếu kiểm tra các biến kiểu số nguyên, số thư. Tuy nhiên có vài trường hợp khi các tham chiếu đối tượng cần được theo dõi để đảm bảo việc phân tán taint hoạt động chính xác.

- Lôgic phân tán taint: Máy ảo Dalvik hoạt động trên cách tập lệnh phân biệt của ngôn ngữ máy DEX, vì thế cần thiết kế một logic phân tán tương ứng [12, tr.5-6]. Và logic luồng dữ liệu đã được chọn, các luồng kiểm tra đều yêu cần xử lý tĩnh nên kéo theo vấn đề hiệu năng và vượt mức kiểm tra dữ đoán. Nên phải bắt đầu bằng việc định nghĩa các đánh dấu taint, các thẻ taint, các biến và việc phân tán taint.
- Đánh dấu các tham chiếu đối tượng: Các luật phát tán trong bảng 2.1 dưới đây rất rõ ràng với 2 ngoại lệ. Thứ nhất, các logic phát tán taint thường đính thẻ taint của một chỉ số mảng trong lúc tìm kiếm để xử lý thông dịch các bảng (ví dụ: ASCII/UNICODE hoặc chuyển đổi ký tự). Ví dụ cân nhắc biên dịch từ các ký tự viết thường sang các ký tự viết hoa nếu một giá trị được đánh dấu "a" dùng như một chỉ số mảng, giá trị "A" trả về được đánh dấu qua giá trị "A" trong mảng là không nên. Vì thế logic taint sẽ dùng cả mảng và taint để đánh chỉ số mảng. Thứ hai là khi mảng chứa các tham chiếu đối tượng (ví dụ: một mảng nguyên), chỉ số thẻ taint được phân tán tới tham chiếu đối tượng và không tới giá trị đối tượng. Vì thế chúng ta đính thẻ taint chứa tham chiếu đối tượng trong luật get của thực thể.

Bảng 2.1: Logic Phân Tán Taint Theo Ngôn Ngữ Máy DEX

Op Format	Op Semantics	Taint Propagation	Description
const-op v_A C	$v_A \leftarrow C$	$\tau(v_A) \leftarrow \emptyset$	Clear v_A taint
move-op v_A v_B	$v_A \leftarrow v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
move-op-R v_A	$v_A \leftarrow R$	$\tau(v_A) \leftarrow \tau(R)$	Set v_A taint to return taint
return-op v_A	$R \leftarrow v_A$	$\tau(R) \leftarrow \tau(v_A)$	Set return taint ($\emptyset$ if void)
move-op-E v_A	$v_A \leftarrow E$	$\tau(v_A) \leftarrow \tau(E)$	Set v_A taint to exception taint
throw-op v_A	$E \leftarrow v_A$	$\tau(E) \leftarrow \tau(v_A)$	Set exception taint
unary-op v_A v_B	$v_A \leftarrow \otimes v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
binary-op v_A v_B v_C	$v_A \leftarrow v_B \otimes v_C$	$\tau(v_A) \leftarrow \tau(v_B) \cup \tau(v_C)$	Set v_A taint to v_B taint $\cup$ v_C taint
binary-op v_A v_B	$v_A \leftarrow v_A \otimes v_B$	$\tau(v_A) \leftarrow \tau(v_A) \cup \tau(v_B)$	Update v_A taint with v_B taint
binary-op v_A v_B C	$v_A \leftarrow v_B \otimes C$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
aput-op v_A v_B v_C	$v_B[v_C] \leftarrow v_A$	$\tau(v_B[\cdot]) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_A)$	Update array v_B taint with v_A taint
aget-op v_A v_B v_C	$v_A \leftarrow v_B[v_C]$	$\tau(v_A) \leftarrow \tau(v_B[\cdot]) \cup \tau(v_C)$	Set v_A taint to array and index taint
sput-op v_A f_B	$f_B \leftarrow v_A$	$\tau(f_B) \leftarrow \tau(v_A)$	Set field f_B taint to v_A taint
sget-op v_A f_B	$v_A \leftarrow f_B$	$\tau(v_A) \leftarrow \tau(f_B)$	Set v_A taint to field f_B taint
iput-op v_A v_B f_C	$v_B(f_C) \leftarrow v_A$	$\tau(v_B(f_C)) \leftarrow \tau(v_A)$	Set field f_C taint to v_A taint
iget-op v_A v_B f_C	$v_A \leftarrow v_B(f_C)$	$\tau(v_A) \leftarrow \tau(v_B(f_C)) \cup \tau(v_B)$	Set v_A taint to field f_C and object reference taint

Đăng ký các biến và các trường của lớp được tham chiếu tương ứng bởi Vx và Fx, R và E là các biến trả lại và ngoại lệ duy trì bên trong bộ biên dịch. A, B và C là các hằng mã byte.

```
public static Integer valueOf(int i) {
  if (i < -128 || i > 127) {
    return new Integer(i); }
  return valueOfCache.CACHE [i+128];
}
static class valueOfCache {
  static final Integer[] CACHE = new Integer[256];
  static {
    for(int i=-128; i<=127; i++) {
      CACHE[i+128] = new Integer(i); } }
}
```

Hình 2.7. Trích đoạn code lấy từ lớp dùng số nguyên của Android

(Minh họa việc cần thiết để phấn tán taint tham chiếu đối tượng)

Đoạn mã nguồn ở hình 2.7 chứng tỏ một thực thể thực thụ của tham chiếu taint là cần thiết. valueOf() trả lại đối tượng số nguyên. Nếu tham số nguyên có giá trị nhỏ hơn -128 hoặc lớn hơn 127, hàm trả lại tham chiếu đến đối tượng nguyên tĩnh, nó được gọi tường minh để chuyển đổi đến một đối tượng.

2.3.3. Phân tán các taint mã nguyên gốc

Mã nguyên gốc (native code) không được theo dõi trong TaintDroid. Lý tưởng là chúng ta có được ngữ nghĩa phân tán như trong bản sao biên dịch. Do đó sẽ cần 2 tiền điều kiện cho việc kiểm tra taint chính xác trong môi trường như Java. TaintDroid có được các tiền điều kiện thông qua việc đo không tự động, hồ sơ phương thức và phụ thuộc vào các yêu cầu theo tình huống.

- Các phương thức máy ảo nội tại (Internal VM Methods): Được gọi trực tiếp bằng mã biên dịch thông qua con trỏ tới mảng các đối số 32bit được đăng ký và con trỏ trả lại giá trị. Sự gia tăng ngăn xếp trong hình 2.6 cung cấp quyền truy cập tới các thẻ taint của cả đối số Java và giá trị trả về. Có 185 phương thức nội tại trong Android 2.1, tuy vậy chỉ có gói được yêu cầu phương thức nguyên gốc System.arraycopy() [12, tr.6-7] dùng cho việc sao chép nội dung mảng và vài phương thức gốc thực thi Java.
- JNI Methods: JNI Methods được gọi thông qua cầu nối lời gọi JNI. Cầu nối lời gọi phân tích các đối số Java và gán giá trị trả về cùng đặc tả chuỗi phương thức. Chúng ta đóng gói cầu nối lời gọi để cung cấp việc phân tán taint cho tất cả các phương thức JNI. Khi một phương thức JNI trả kết quả, TaintDroid tham khảo bảng hồ sơ phương thức cho việc cập nhật phân tán thẻ. Một hồ sơ phương

thức là một danh sách từ/tới các cặp chỉ ra các luồng giữa các biến, nó có thể là các đối số phương thức, các biến của lớp hoặc các giá trị trả về. Liệt kê các luồng thông tin cho tất cả các phương thức JNI như việc tiêu tốn thời gian tốt nhất để tự động dùng xử lý mã nguồn.

2.3.4. Phân tán taint IPC

Các thẻ taint phải được phân tán giữa các ứng dụng khi chúng trao đổi dữ liệu. Việc theo dõi ảnh hưởng đến hiệu năng và bộ nhớ hệ thống. TaintDroid theo dõi taint ở mức thông điệp. Một thẻ taint thông điệp thể hiện bên trên của các đánh dấu taint được gán cho các biến chứa bên trong thông điệp. Chúng ta dùng tính chất của mức thông điệp để tối ưu hiệu năng và lưu trữ trong lúc IPC. Và cũng chọn việc thực hiện mức thông điệp trên việc phân tán taint mức biến vì trong một hệ thống mức biến, một bộ nhận có thể kiểm soát bởi việc mở gói các biến theo cách khác để biết thông tin không cần phân tán taint.

2.3.5. Phân tán các taint thiết bị lưu trữ thứ cấp

Các thẻ taint có thể bị mất khi lưu dữ liệu vào tệp, thiết kế hiện tại lưu mỗi thẻ taint vào một tệp. Để làm điều này chúng ta đã phải mở rộng thuộc tính hỗ trợ bởi hệ thống tệp của máy chủ Android và định dạng thẻ nhớ ngoài theo hệ thống tệp ext2. Như với các mảng và IPC, lưu trữ mỗi thẻ taint mỗi tệp dẫn đến vấn đề giới hạn của các đánh dấu taint cho CSDL thông tin. Cách khác chúng ta có thể kiểm tra các thẻ taint bằng một bộ tinh chỉnh để giảm chi phí của việc thêm bộ nhớ và vấn đề hiệu năng.

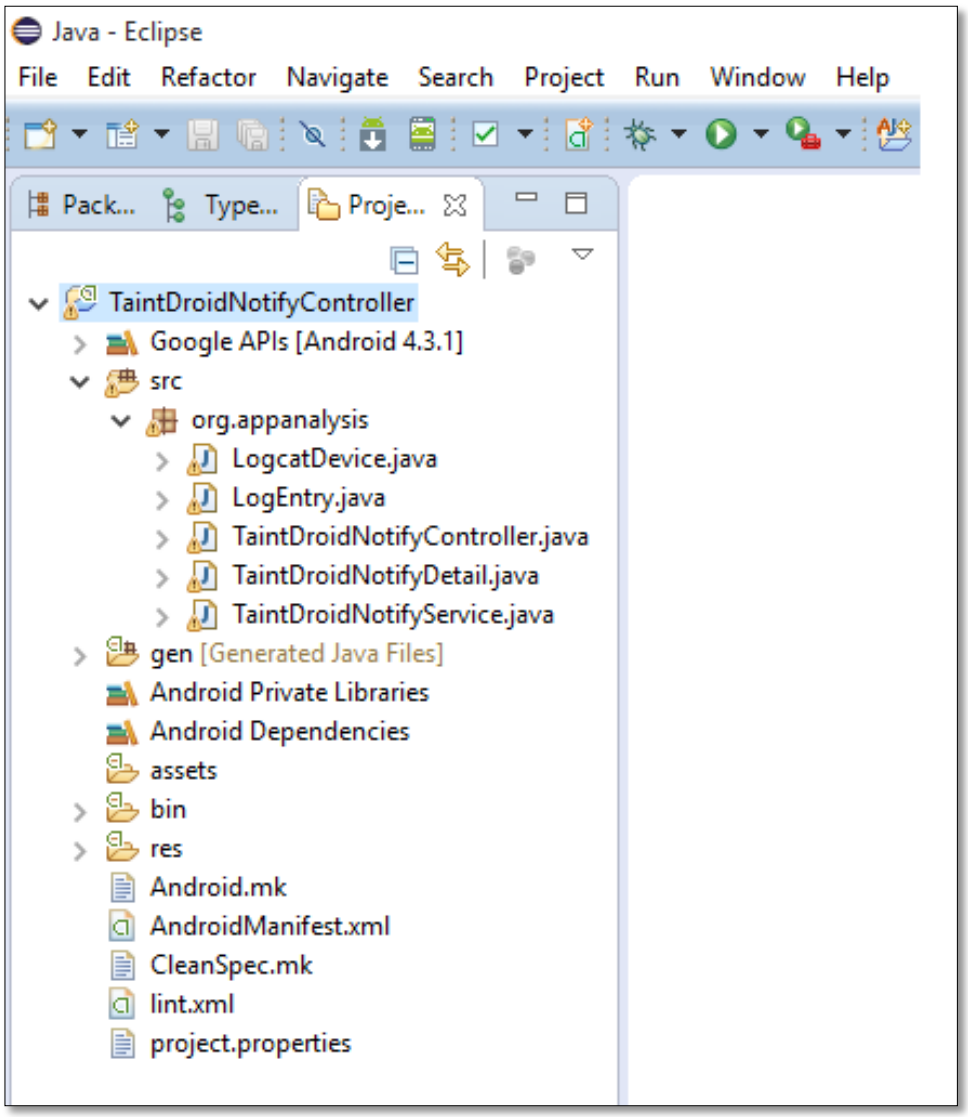
2.3.6. Thư viện giao diện taint

Tài nguyên taint được định nghĩa bên trong môi trường ảo hóa phải kết nối các thẻ taint với hệ thống kiểm tra. Chúng ta trừu tượng tài nguyên taint vào một thư viện giao diện taint đơn lẻ. Giao diện thực hiện 2 chức năng: 1) thêm các đánh dấu taint vào các biến; và 2) nhận các đánh dấu taint từ các biến. Thư viện chỉ cung cấp khả năng để thêm, bớt các thẻ taint như chức năng có thể được dùng bởi mã Java không tin cậy để xóa các đánh dấu taint.

Thêm các thẻ taint vào các mảng và các chuỗi qua các phương thức nội tại của máy ảo là minh bạch, cả hai lưu trong các đối tượng dữ liệu. Các biến kiểu nguyên gốc, một cách khác được lưu trữ trong các ngăn xếp của trình biên dịch và không xuất hiện dưới dạng lời gọi hàm. Tuy nhiên thư viện taint dùng giá trị trả về như các biến kiểu nguyên gốc. Lập trình viên đưa một giá trị hoặc biến thích hợp taint vào phương thức (ví dụ: addTaintInt()) và biến trả lại có được giá trình tương tự việc thêm sẽ xác định được thẻ taint. Chú ý bộ nhớ ngăn xếp không quản lý cho việc nhận thẻ taint.

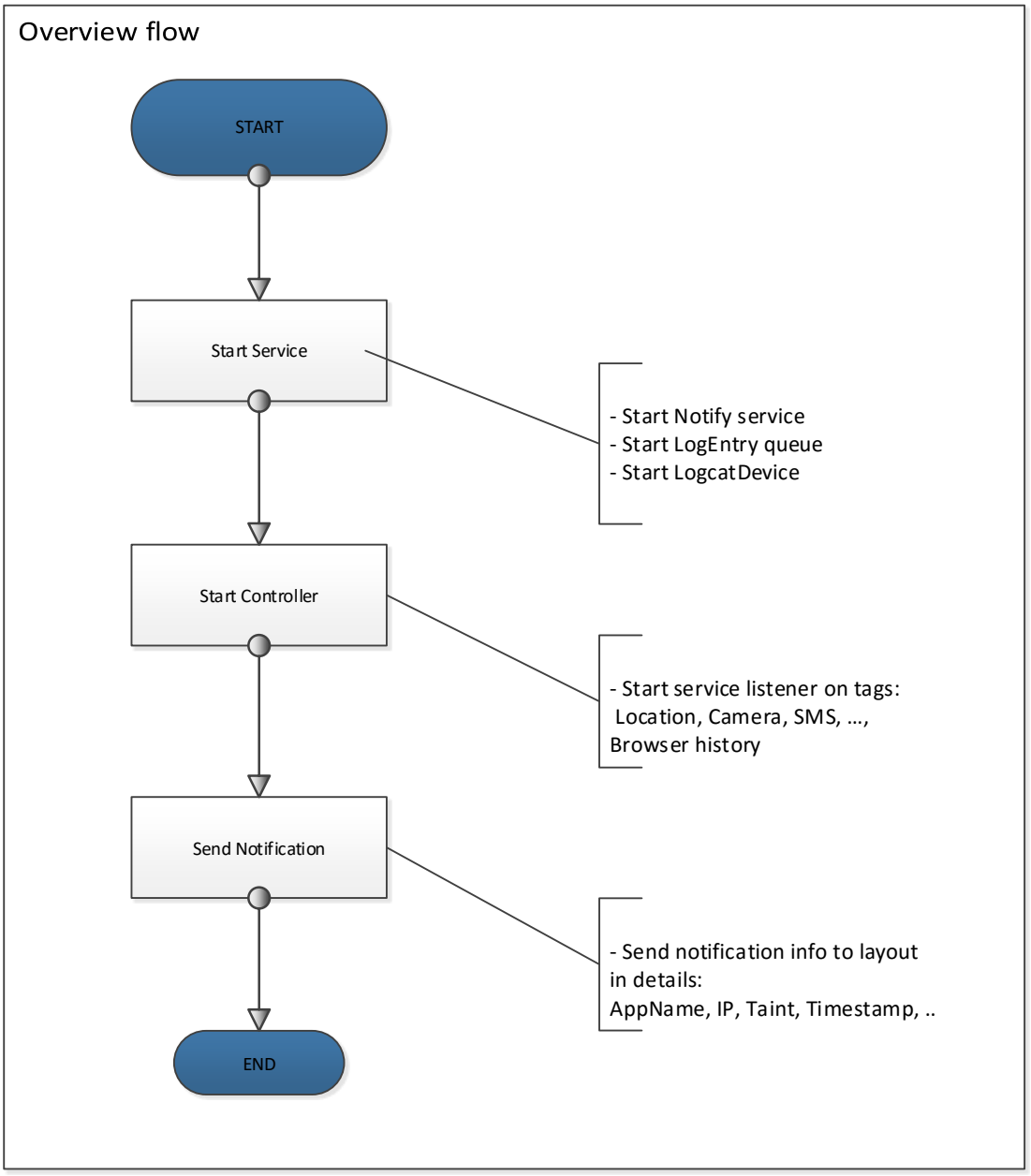
2.4. Phân tích hệ thống hiện tại

Như đã giới thiệu ở trên, TaintDroid là sự mở rộng của HĐH Android. Nó điều chỉnh để hỗ trợ việc theo dõi luồng dữ liệu ở mức hệ thống và theo dõi việc truy cập tài nguyên của hệ thống thông qua các taint. Việc kiểm tra, quản lý và thông báo của hệ thống thông qua chương trình Taintdroid Notify Controller. Dưới đây là cấu trúc chương trình và miêu tả chung về từng lớp của mã nguồn hiện tại.



Hình 2.8. Cấu Trúc Android Notify Controller

Phần trên đã miêu tả đầy đủ cấu trúc chương trình thực thi dưới dạng các gói, lớp mã nguồn tương ứng với từng mô đun (module). Và dưới đây miêu tả chi tiết luồng xử lý của chương trình và đặt tả chức năng của từng module.



Hình 2.9. Luồng Xử Lý Android Notify Controller

- LogEntry: LogEntry là lớp chứa thông tin chi tiết của một log của chương trình, mỗi logEntry gồm có các thông tin như: thời gian, thẻ, chỉ số tiến trình và thông điệp.

- LogcatDevice: LogcatDevice là lớp thực hiện việc quản lý việc đọc ghi log của chương trình. Khi bắt đầu việc ghi một log, một tiến trình sẽ được tạo lập và lưu các log entry vào trong tiến trình này. Nó cũng hỗ trợ việc các phương thức để đóng, mở tiến trình cũng như đọc thông tin logEntry. Thực chất việc đọc ghi trên logEntry trong chương trình là đọc ghi trên bộ nhớ đệm của tiến trình.

- TaintDroidNotifyDetail: Lớp này được mở rộng từ lớp Activity của framework Android, nó lưu chi tiết thông tin của một thẻ thông báo và lớp gồm có các đối tượng sau:

- AppName: Tên ứng dụng được xử lý
- IpAddress: Địa chỉ IP được ứng dụng gửi tới, thường là địa chỉ máy chủ quảng cáo, máy chủ lưu thông tin nhạy cảm lấy được
- Taint: Mã kiểm tra của TaintDroid
- Data: Thông tin chi tiết của một thẻ thông báo
- Id: Định danh của thông báo
- TimeSpamp: Thời gian ghi nhận thông tin

- TaintDroidNotifyService: Lớp này được mở rộng từ lớp Service của framework Android, nó quản lý việc chạy TaintDroid Notify dưới dạng một dịch vụ. Nó quản lý việc chạy dịch vụ, lập hàng đợi và cũng tra ra thông tin chi tiết một thực thể (instance) của một TaintDroidNotifyDetail. tTable là bảng định nghĩa các taint dưới dạng bảng băm, cặp <chỉ số, đánh dấu> tương ứng với các taint sẽ được xử lý và dưới đây là danh sách các cặp taint và đánh dấu:

Chỉ số	**Đánh dấu**
0x00000001	Location
0x00000002	Address Book (ContactsProvider)
0x00000004	Microphone Input
0x00000008	Phone Number
0x00000010	GPS Location
0x00000020	NET-based Location
0x00000040	Last known Location
0x00000080	Camera
0x00000100	Accelerometer
0x00000200	SMS
0x00000400	IMEI
0x00000800	IMSI
0x00001000	ICCID (SIM card identifier)
0x00002000	Device serial number
0x00004000	User account information
0x00008000	Browser history

Hiện tại taint của lịch sử trình duyệt (Browser history) đã được ghi lại nhưng chưa được phân tích và thông báo cho người dùng. Trong khuôn khổ để tài sẽ tập trung phân tích và xử lý thông tin Browser history và thông báo cho người dùng.

- Starter: Là một lớp phụ mở rộng từ lớp BroadcastReceiver của framwork Android thực hiện việc chạy dịch vụ khi có yêu cầu.
- Producer: Là một lớp phụ thực thi giao diện Runnable của framwork Android, khi chạy sẽ xử lý việc đọc ghi log của LogcatDevice và quản lý theo hàng đợi.
- Consumer: Là một lớp phụ thực thi giao diện Runnable của framwork Android, khi chạy sẽ xử lý các LogEntry theo hàng đợi. Mỗi lần xử lý thành công nó sẽ gửi thông báo tới người dùng dưới dạng Notification trên thanh trạng thái của thiết bị.

▪ get_Processname: Là một phương thức riêng (private) của lớp trả lại tên của tiến trình. Khi thực thi nó sẽ quét từ danh sách các tiến trình đang chạy và tìm tiến theo định danh.

▪ get_taint: Là một phương thức riêng của lớp trả lại taint dưới dạng chuỗi theo thông điệp truyền vào. Dựa vào bảng băm tTable đã định nghĩa để tìm ra taint tương ứng.

▪ sendTaintDroidNotification: Là phương thức thực hiện việc gửi thông báo tới thiết bị, nó sẽ tạo ra một đối tượng thông báo theo kiểu BigTextStyle của Android. Sau đó sẽ gán các trường (appName, IP, Data, …) vào thông báo rồi gửi đến thiết bị thông qua lớp NotificationManager của Android. Ngoài ra khi dịch vụ bắt đầu được chạy, dịch vụ LogcatDevice cũng được chạy và thực thể của 2 lớp Proceducer, Consumer cũng được chạy tướng ứng với 2 luồng riêng thông qua việc ghi đè phương thức onStartCommand() của lớp Service. Khi tắt dịch vụ thì đối tượng trên cũng được đóng hay xóa thông qua việc ghi đè phương thức onDestroy() của lớp Service.

- TaintDroidNotifyController**:** Controller này là lớp cũng được mở rộng từ lớp Activity của framework Android. Nó chuyển tiếp các hành động Start/Stop dịch vụ của chương trình. Đây là lớp sau của giao diện chương trình Controller, khi được khởi tạo nó sẽ kích hoạt các nút Start/Stop của ứng dụng và ở trang thái chờ người dùng yêu cầu. TaintDroid có một ứng dụng để hỗ trợ người dùng kích hoạt dịch vụ của nó (TaintDroid Notification Service). Khi dịch vụ được chạy, TaintDroid bắt đầu theo dõi việc truy cập các dữ liệu nhạy cảm và thông báo cho người sử dụng. Ứng dụng có tên TaintDroid Notifiy, nó có giao diện khá đơn giản gồm 2 phần chính như sau:

- Màn hình hiện thị chức năng bật và tắt dịch vụ tướng ứng với 2 nút Start và Stop.
- Màn hình hiển thị thông tin thông báo truy cập dữ liệu nhạy cảm. Khi TaintDroid phát hiện thông tin nhạy cảm được truy cập hay gửi đến máy chủ từ xa, nó sẽ hiện thông báo dưới dạng thông báo hệ thống. Khi người dùng mở thông báo thì màn hình này sẽ hiện các thông tin chi tiết của thông báo. Giao diện chương trình như hình dưới đây là giao diện của TaintDroid để người dùng có thể khởi động hay dừng chương trình. Nó được thông qua một module có tên gọi TaintDroid Notify Controlller.

Hình 2.10. Giao Diện TaintDroid Notify Controlller

2.5. Đánh giá hiệu năng

Dựa trên kết quả khảo sát 50 ứng dụng miễn phí phổ biến trong mỗi lĩnh vực trên kho ứng dụng Android (Android Market). Trong đó có 1/3 ứng dụng yêu cầu quyền truy nhập Internet kèm theo quyền truy cập ví trí, camera hoặc dữ liệu âm thanh. Từ tập ứng dụng trên, chọn ngẫu nhiên 30 ứng dụng phổ biến. Bảng 2.2 liệt kê các ứng dụng với các quyền được yêu cầu khi cài đặt lần đầu. Điều

này không phản ánh thực tế việc truy cập hay sử dụng dữ liệu nhạy cảm mà chúng có thể sẽ truy xuất.

Hệ thống TaintDroid đã lưu log chi tiết thông tin như: thông điệp taint, tệp taint đầu ra và thông điệp mạng taint với địa chỉ từ xa. Trong vòng 100 phút thử nghiệm, có tầm 22.594 gói tin (8.6MB) và 1.130 kết nối TCP. Hơn nữa khi kiểm tra đường truyền mạng, các ứng dụng đã được người dùng vô tình hay cố ý chấp nhận truyền dữ liệu nhạy cảm ra bên ngoài. Điều này cung cấp thêm thông tin để xác định có thể có sự vi phạm chính sách. Ví dụ với việc chọn "dùng vị trí của tôi" trong ứng dụng thời tiết, người dùng đã đồng ý tiết lộ tọa độ địa lý với máy chủ thời tiết.

Bảng 2.2 : Ứng dụng được nhóm theo quyền đã yêu cầu (L: location, C: camera, A: audio, P: phone state)

Applications*	#	Permissions†			
		L	C	A	P
The Weather Channel (News & Weather); Cestos, Solitaire (Game); Movies (Entertainment); Babble (Social); Manga Browser (Comics)	6	x			
Bump, Wertago (Social); Antivirus (Communication); ABC — Animals, Traffic Jam, Hearts, Blackjack, (Games); Horoscope (Lifestyle); Yellow Pages (Reference); 3001 Wisdom Quotes Lite, Dastelefonbuch, Astrid (Productivity), BBC News Live Stream (News & Weather); Ringtones (Entertainment)	14	x			x
Layar (Lifestyle); Knocking (Social); Coupons (Shopping); Trapster (Travel); Spongebob Slide (Game); ProBasketBall (Sports)	6	x	x		x
MySpace (Social); Barcode Scanner, ixMAT (Shopping)	3		x		
Evernote (Productivity)	1	x	x	x	

Bảng 2.3 : Các vi phạm tiềm ẩn trong 20 ứng dụng thử nghiệm

Observed Behavior (# of apps)	Details
Phone Information to Content Servers (2)	2 apps sent out the phone number, IMSI, and ICC-ID along with the geo-coordinates to the app's content server.
Device ID to Content Servers (7)*	2 Social, 1 Shopping, 1 Reference and three other apps transmitted the IMEI number to the app's content server.
Location to Advertisement Servers (15)	5 apps sent geo-coordinates to ad.qwapi.com, 5 apps to admob.com, 2 apps to ads.mobclix.com (1 sent location both to admob.com and ads.mobclix.com) and 4 apps sent location† to data.flurry.com.

Bảng 2.3 đã tóm tắt kết quả mà TaintDroid đã đánh dấu 105 kết nối TCP có chứa thông tin nhạy cảm. Hệ thống đã sử dụng các tên từ xa như các chỉ dẫn về nơi dữ liệu bị gửi đi đến máy chủ của bên thứ 3. Nó chỉ ra 21 trong số 30 ứng dụng yêu cầu quyền đọc trạng thái điện thoại và kết nối Internet. Hệ thống đã tìm ra 2 trong số 21 ứng dụng đã truyền số điện thoại đến máy chủ, số IMEI cũng bị 9 ứng dụng gửi đi.

Thông tin vị trí bị truyền đến máy chủ quảng cáo, ½ ứng dụng đã truyền thông tin vị trí đến máy chủ quảng cáo mà không có có sự chấp nhận của người dùng. Ví dụ về dữ liệu vị trí được truyền đến AdMob dưới dạng chuỗi như sau:

...&s=a14a4a93f1e4c68&..&t=062A1CB1D476DE85
B717D9195A6722A9&d%5Bcoord%5D=47.6612278900
00006%2C-122.31589477&...

Theo tìm hiểu từ AdMod SDK thì tham số s = ? là chỉ định danh của ứng dụng phát hành, coord = ? là tọa độ địa lý của thiết bị.

Chương 3. Cải tiến theo dõi nguồn dữ liệu nhạy cảm

Trước khi miêu tả chi tiết các giải pháp cải tiến theo dõi dữ liệu nhạy cảm, chúng ta sẽ phân tích chi tiết các thành phần của hệ thống hiện tại. TaintDroid chỉ kiểm tra ở mức luồng dữ liệu, không thể kiểm tra ở mức luồng điều khiển để có thể tối ưu hóa hiệu năng. TaintDroid chỉ có thể kiểm tra dữ liệu nhận được của ứng dụng và những hành động khả nghi. Tuy nhiên các ứng dụng độc hại thực thụ nó sẽ lấy thông tin của người dùng thông qua các luồng điều khiển. Việc kiểm tra luồng điều khiển yêu cầu việc xử lý tĩnh, có nghĩa không thể xử lý các ứng dụng bên thứ 3 khi không có mã nguồn. Điều khiển trực tiếp các luồng có thể kiểm tra được một các linh hoạt khi taint được xác định. Tuy nhiên DEX không duy trì các cấu trúc nhánh để TaintDroid có thể dựa vào. Theo yêu cầu của xử lý tĩnh thì phải xác định được các biểu đồ luồng điều khiển (CFGs) phương thức [12, tr.12]. Tuy nhiên TaintDroid hiện không thực hiện xử lý như vậy để tránh tốn chi phí hiệu năng. Cuối cùng khi thông tin được gửi đi từ điện thoại, nó có thể trả lại bên trong một tín hiện reply mạng và nó không thể kiểm tra được thông tin loại này. Ví dụ về hạn chế của việc kiểm tra ở mức luồng dữ liệu, nếu ứng dụng A mã hóa dữ liệu trước khi truyền đi đến máy chủ. Nó không thể kiểm tra và phát hiện taint đó có sử dụng dữ liệu nhạy cảm của người dùng hay không và sẽ bỏ qua dữ liệu đó.

- Hạn chế thực thi: Android dùng thực thi kết hợp Apache của Java với một số sửa đổi tùy biến. Việc thực thi này gồm sự hỗ trợ cho lớp PlatformAddress, nó gồm một địa chỉ nguyên gốc và dùng bởi các đối tượng bộ nhớ đệm trực tiếp (DirectBuffer). Tệp và các hàm thư viện vào ra đọc và ghi trực tiếp các biến cho địa chỉ nguyên gốc từ một DirectBuffer. TaintDroid không kiểm tra các thẻ taint trên các đối tượng DirectBuffer vì dữ liệu được lưu trong các cấu trúc dữ liệu nguyên thủy và nó lưu các log khi đọc hay ghi biến trực tiếp.

Hạn chế tài nguyên Taint: Khi TaintDroid kiểm tra thông tin nhạy cảm, nó làm mất khả năng kiểm tra thông tin tồn tại trong các định danh cấu hình. Ví dụ chuỗi số IMSI gồm có mã nước (MCC – Mobile Country Code), mã mạng di động (MNC - Mobile Network Code) và số định danh trạm di động (MSIN – Mobile Station Identifier Number) [12, tr.12], chúng đều được đánh dấu cùng nhau. Nó dùng MCC và MNC để mở rộng các các tham số cấu hình khi kết nối các dữ liệu khác. Điều này làm tất cả các thông tin trong gói được đánh dấu. Như vậy với các tài nguyên taint chứa các tham số cấu hình, các biến đánh dấu

riêng với gói trở lên phù hợp hơn. Tuy nhiên như kết quả đã phân tích, việc theo dõi ở mức thông điệp làm ảnh hưởng đến các tài nguyên.

Việc cấu hình và cài đặt TaintDroid cũng có hạn chế do hệ thống không phải là một module thiết kế hoạt động trên HĐH Android. Nó thực chất là một HĐH Android sửa đổi để phục vụ được hoạt động kiểm tra luồng dữ liệu cũng như cài đặt một ứng dụng chạy dưới dang dịch vụ như Google Play Service. Điều này có nghĩa người phát triển phải có kiến thức vững vàng trong cài đặt cấu hình để xây dựng được HĐH Android. Để xây được TaintDroid người dùng phải tải toàn bộ mã nguồn của nó tương đương với mã nguồn HĐH Android với dung lượng lớn. Trước đó phải cài đặt một hệ thống tương thích để xây Android. Hệ thống cũng phải cập nhật đầy đủ các bản vá cũng như các cập nhật cần thiết theo yêu cầu phát triển ứng dụng Android khuyến nghị bởi website android source [3]. Khi cài đặt hệ thống rất dễ gây ra lỗi biên dịch, mỗi hành động cần được sao lưu kịp thời để tránh mất các cài đặt thành công trước đó.

Biên dịch TaintDroid cũng mất nhiều thời gian và tương đương như biên dịch HĐH Android. Mỗi thay đổi nhỏ từ ứng dụng thực thi của TaintDroid đều phải biên dịch lại toàn bộ hệ thống. Mỗi lần như thế hệ thống đều phải tiến hành xóa các thông số trước biên dịch và tạo lại các tệp ảnh (boot.img, system.img, …).

Khi so sánh TaintDroid với một số hệ thống an ninh di động khác như Samsung Knox, BlackBerry Balance thì nó có một số ưu điểm như: Có thể chạy ngay trong HĐH và xử lý dữ liệu theo luồng, hỗ trợ phiên bản cũ của HĐH Android, không tốn chi phí triển khai. Tuy nhiên vẫn còn một số nhược điểm như không bảo mật được nhiều lớp và hỗ trợ bảo mật phần cứng như Samsung Knox, không hỗ trợ phân vùng an toàn. BlackBerry Balance có thể hỗ trợ quản lý dữ liệu cá nhân riêng biệt và có mã hóa. Để dẫn chứng về hạn chế của TaintDroid, ScrubDroid đã đưa ra một chứng minh có thể dễ dàng thực hiện các kiểu tấn công gián tiếp bởi các con trỏ điều khiển. Quá trình này sử dụng giao diện vào ra Java của Android để nhận được truy cập bộ nhớ trực tiếp. Tấn công này cũng khó có thể đi lệch hướng trong việc lưu các đánh dấu taint cho mỗi byte bộ nhớ. Nhóm tấn công tin rằng các lớp của phương thức có thể được nhìn thấy bởi các thực thi lệnh được ủy quyền cho các thành phần khác của hệ thống. Với GPUs việc tín toán trở nên mạnh mẽ hơn, các phần mềm độc hại có thể gỡ bỏ các ủy quyền của các dấu taint để đồ thị hóa, hơn nữa việc thực hiện tác vụ này là trực tiếp trên CPU.

Để phát triển ứng dụng cho Android, có thể sử dụng các PC, laptop sử dụng HĐH Windows, MacOS, Ubuntu và dưới đây là cấu hình cho PC, laptop chạy HĐH Windows 7 trở lên. Chi tiết cấu hình cho môi trường phát triển sẽ được miêu tả chi tiết trong phần phụ lục và dưới đây là miêu tả chung.

- Cấu hình máy tính tối thiểu: 4GB RAM, CPU Intel Core I3, 120GB HDD trống.
- Môi trường phát triển Eclipe hoặc Android Studio cho HĐH Windows.
- Yêu cầu cho Eclipe:
  - Eclipe Mar 2.0 trở lên.
  - Android SDK Tools for Windows 20.xx trở lên.
  - Android Development Tools for Eclipe 20.xx trở lên
  - Eclipe IDE for Java Developers 4.3 trở lên.
  - Android Virtual Devices (AVDs).
- Yêu cầu cho Android Studio:
  - Android Studio 2.0 trở lên.
  - Android SDK Tools for Windows 20.xx trở lên.
  - Android SDK Platform 20.xx trở lên..
  - Intel or ARM Sytem Images.
  - Android Virtual Devices (AVDs).

3.1. Giải pháp cải tiến theo dõi truy cập lịch sử trình duyệt

Ở phần trên đã phân tích cụ thể về các ưu điểm và hạn chế của TaintDroid. Qua quá trình nghiên cứu về hệ thống nhận thấy TaintDroid chỉ kiểm tra ở mức các luồng dữ liệu để tối ưu hóa hiệu năng hệ thống. Tuy nhiên các ứng dụng có thể gây hại cho hệ thống thông qua các luồng điều kiển. Vậy cải tiến lớn nhất của hệ thống là làm sao có thể kiểm tra các taint ở mức các luồng điều khiển. Khi đó hệ thống sẽ giám sát được các ứng dụng được linh hoạt và chặc chẽ hơn. Kiểm tra thẻ taint trong đối tượng bộ đệm trực tiếp: TaintDroid thực thi trên Apache của Java, nó chứa các địa chỉ và dùng các đối tượng bộ đệm (Direct-Buffter object). Các ứng dụng có thể sử dụng và thay đổi địa chỉ thông qua các đối tượng trên. Vậy việc kiểm tra được thẻ taint trong các đối tượng đệm trên là một cải tiến không nhỏ nhằm tăng cường khả năng giám sát các thẻ taint của hệ thống. TaintDroid hiện tại có thể kiểm soát nhiều loại thông tin nhạy cảm của người dùng thiết bị như:

- Vị trí
- Vị trí GPS

- Số điện thoại
- Máy ảnh
- Tin nhắn
- Số IMEI
- Số sê ri thiết bị
- …

Nhưng với nhu cầu và thói quen của người dùng TBDĐ hiện nay, việc truy cập Internet là rất cao, cùng với đó là việc sử dụng các ứng dụng truy cập Internet như FaceBook, Twitter, Browser, … để làm việc và giải trí. Như chúng ta đã thấy, với cơ chế hoạt động của trình duyệt website cũng như các ứng dụng hoạt động trên nền website thì có rất nhiều thông tin được trình duyệt lưu lại trong quá trình sử dụng và trong đó cũng chứa nhiều thông tin nhạy cảm.

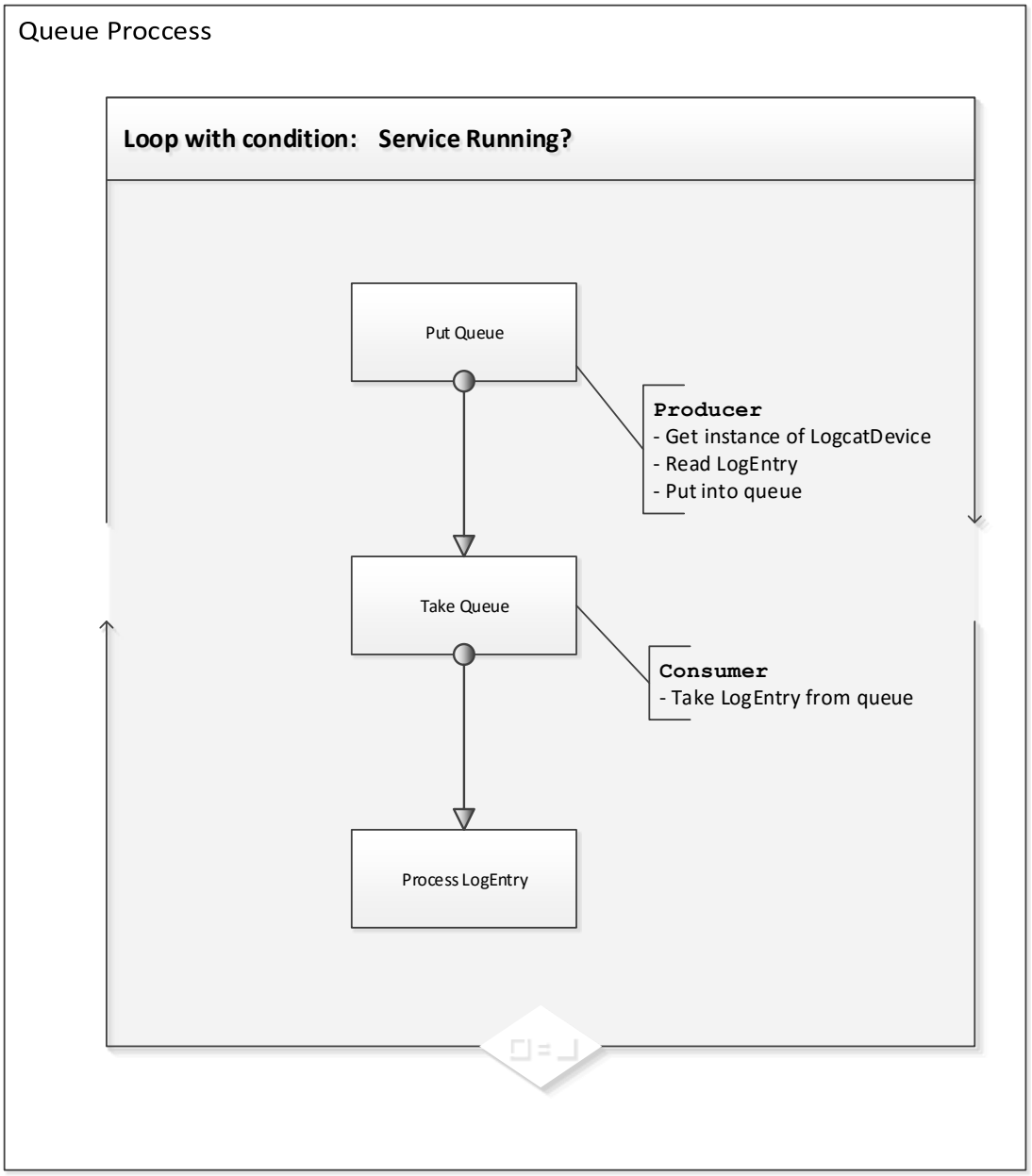
Vậy chúng ta có thể cải tiến hệ thống để hỗ trợ việc kiểm soát truy cập lịch sử của trình duyệt, nó cũng góp phần cho hệ thống được hoàn thiện hơn trong việc giám sát các ứng dụng truy cập dữ liệu nhạy cảm của người dùng. Trong khuôn khổ của việc nghiên cứu trên nền tảng Taintdroid, mọi nghiên cứu cũng như kiểm soát truy cập sẽ được thực hiện trên trình duyệt Google Chrome vì nó là trình duyệt ngầm định trên HĐH Android.

Như đã trình bày, việc cải tiến là nhằm đưa phân tích chi tiết thông tin lịch sử trình duyệt để TaintDroid có thể theo dõi và thông báo khi có truy cập không an toàn. Nên trước tiên sẽ phải thêm các logic trong lớp TaintDroid NotifyService của chương trình TaintDroidNotifyController và xử lý taint này và dưới đây là chi tiết từng bước cần thực hiện giải pháp cái tiến:

- Bước 1: Lấy thông tin log của hệ thống
- Bước 2: Tìm kiếm taint lịch sử trình duyệt
- Bước 3: Xử lý taint
- Bước 4: Điều chỉnh và gửi thông báo

3.1.1. Lấy thông tin log của hệ thống

Lấy thông tin log của hệ thống, để lấy được thông tin log của hệ thống, cần phải xử lý dữ liệu trên lớp TaintDroidNotifyService, đây là lớp thực thi dịch vụ. Mỗi khi thực thi dịch vụ, các block được xử lý liên tục đến khi dừng dịch vụ. Nó hỗ trợ lấy các khối hàng đợi của hệ thống thông qua giao diện BlockingQueue. Các block này sẽ được chuyển đổi sang các đối tượng log tương tự khái báo của TaintDroid. Mỗi khối trong hàng đợi đều chứa đầy đủ các thông tin như: thời gian, mã số tiến trình, thẻ và thông điệp.

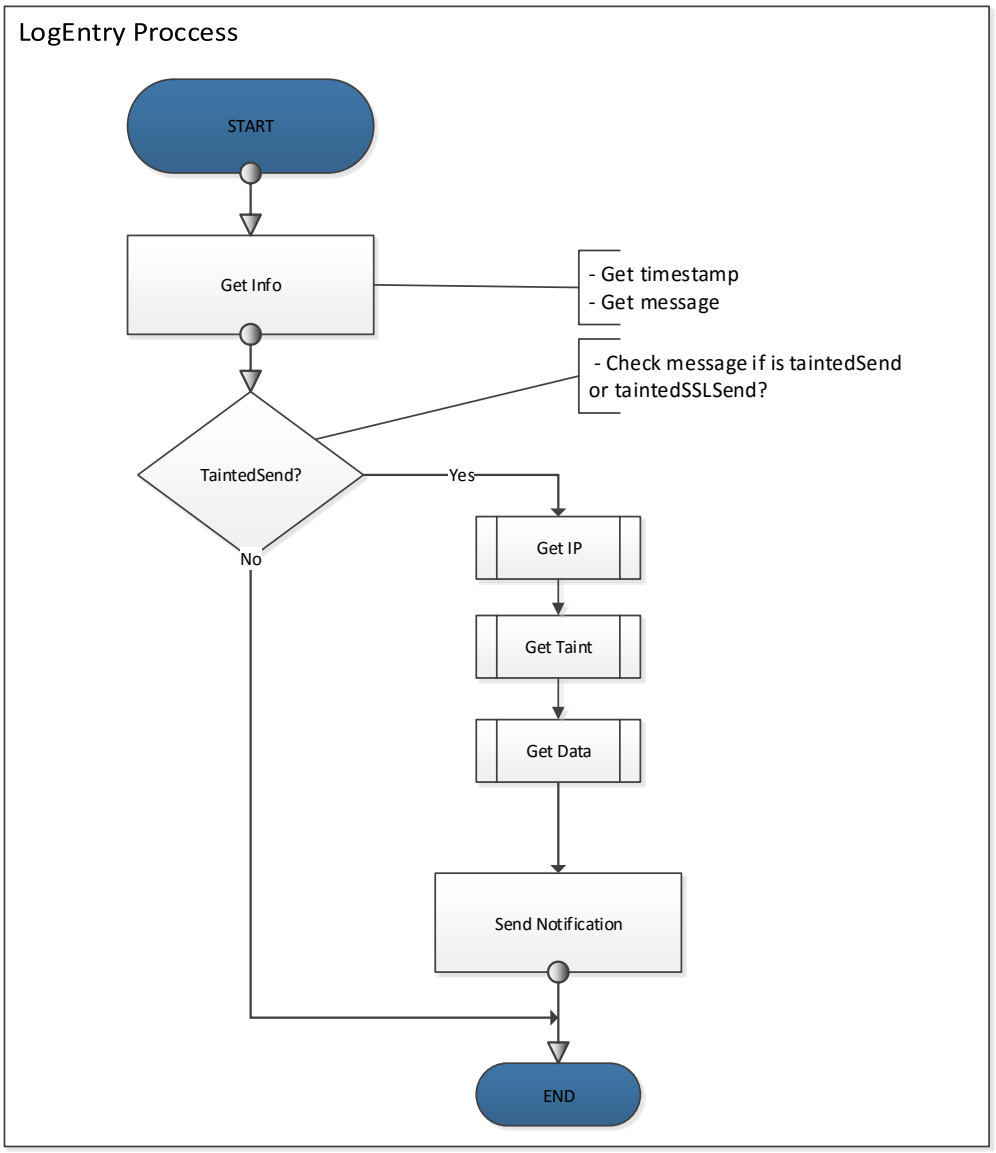


Hình 3.1. Luồng Xử Lý Hàng Đợi

3.1.2. Tìm kiếm taint lịch sử trình duyệt

Tìm kiếm taint lịch sử trình duyệt trên log, khi xử lý từng khối của hàng đợi, hệ thống sẽ trích xuất dữ liệu từ thông điệp để lấy ra các thông tin cần thiết như: địa chỉ IP, kiểu Taint, tên tiến trình, dữ liệu. Cụ thể việc tìm kiếm và định danh taint từ thông điệp là thông qua việc phân tích thông điệp theo mẫu. Nếu tìm được các số của taint thì sẽ so sánh với bảng ánh xạ định nghĩa sẵn. Tại thời điểm trên, chúng ta sẽ tìm các taint tương ứng của browser history.

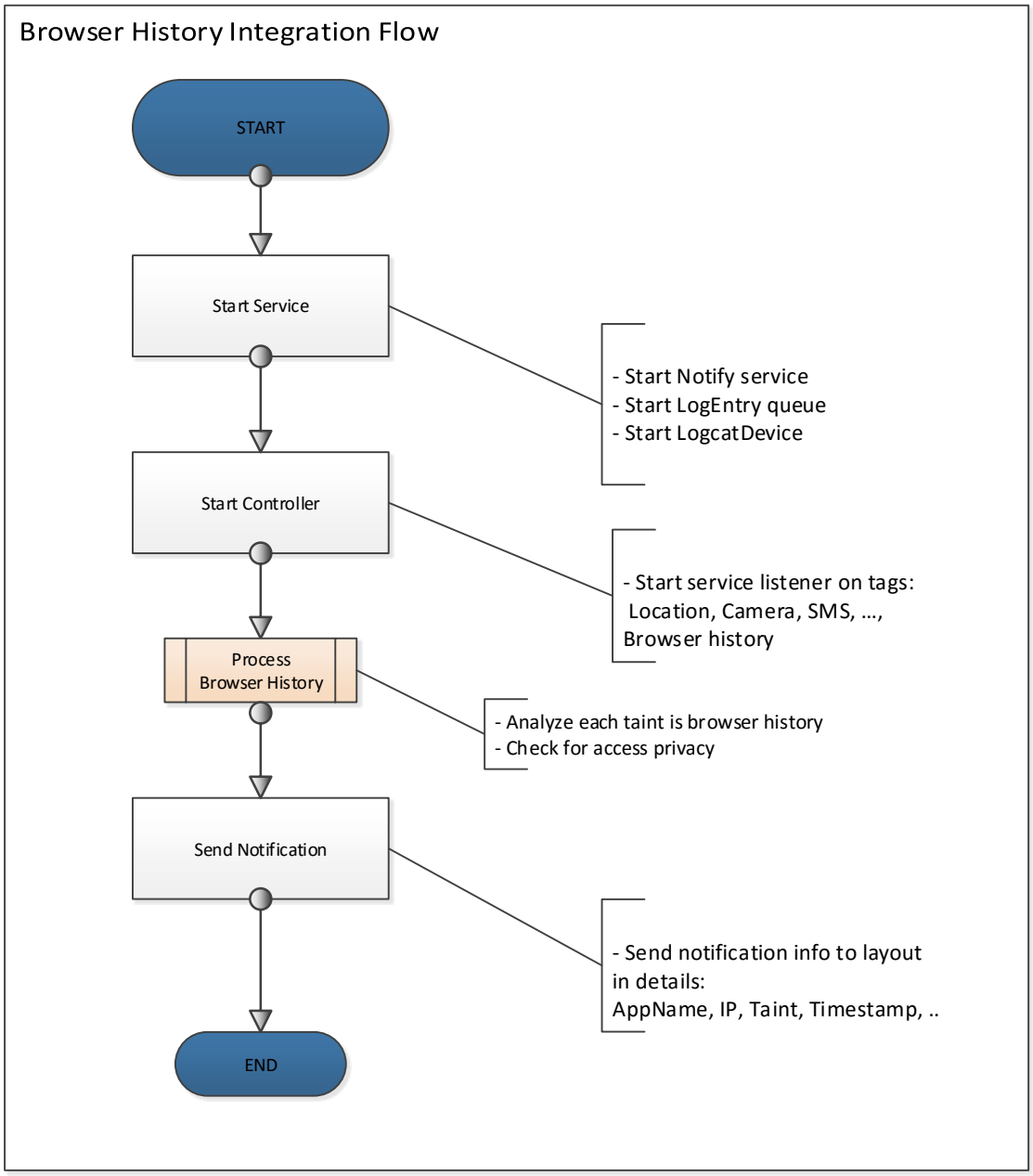
Quá trình tìm kiếm taint thông qua việc xử lý LogEntry được gọi từ luồng xử lý hàng đợi như hình 3.1 ở trên, mỗi khi hàng đợi có thông tin thì sẽ gửi yêu cầu xử lý và gọi hàm processLogEntry(le). Trong đó đối số le là một LogEntry được lấy từ hàng đợi và hình 3.2 dưới đây mô tả sơ đồ luồng xử lý một LogEntry.



Hình 3.2. Luồng Xử Lý LogEntry

3.1.3. Xử lý taint

Trong luồng xử lý taint của hệ thống, chúng ta sẽ tiến hành kiểm tra và xử lý taint lịch sử trình duyệt. Dựa vào dữ liệu taint, chúng ta sẽ tiến hành kiểm tra xem ứng dụng có trong danh sách đen truy cập không. Nếu không thuộc danh sách đen sẽ tiến hành kiểm tra xem có vi phạm truy cập thông tin đã định nghĩa. Hình 3.3 dưới đây sẽ miêu tả chi tiết luồng xử lý lịch sự trình duyệt tích hợp trong luồng xử lý của hệ thống.

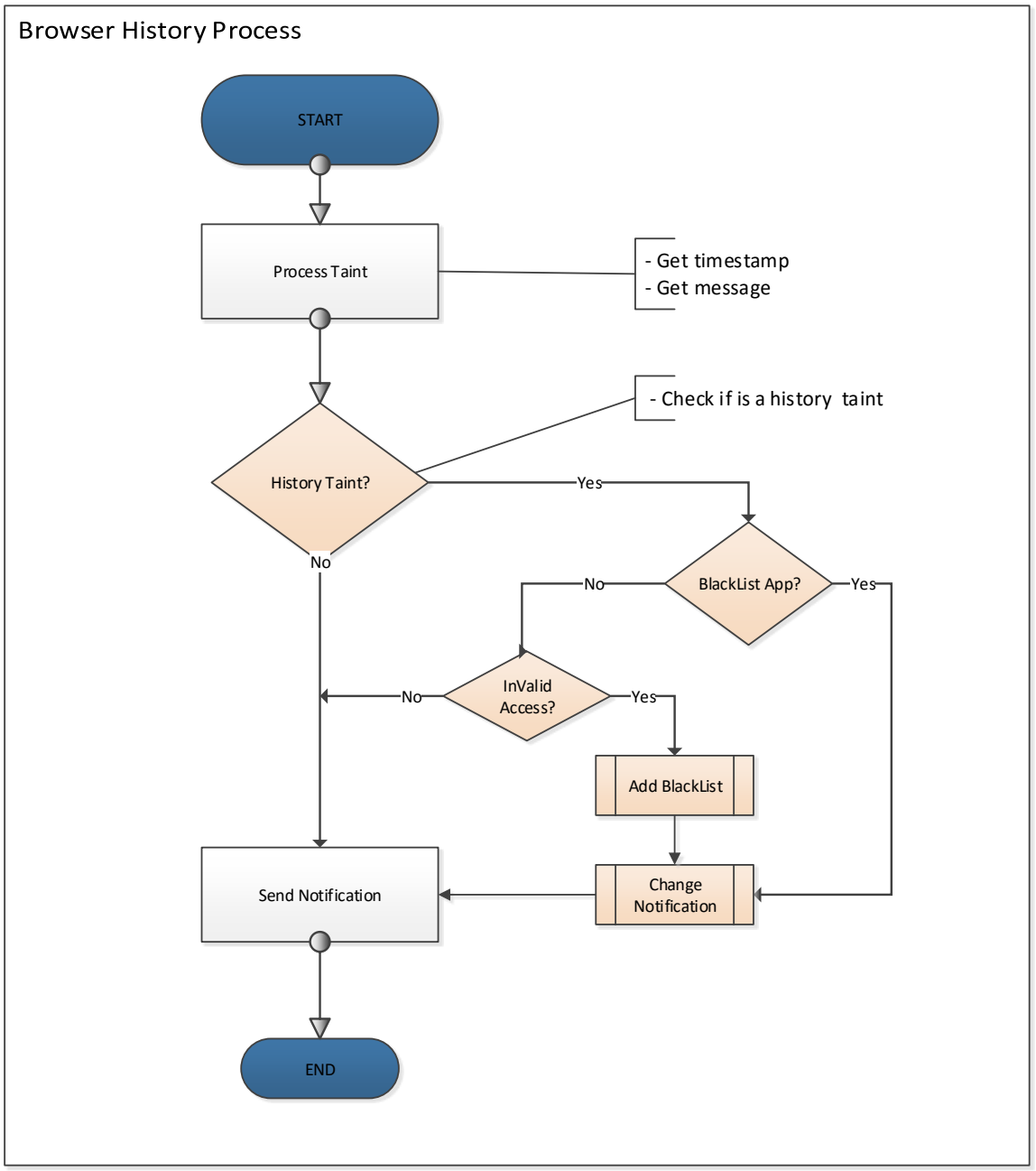


Hình 3.3. Luồng Xử Lý Browser History Tích Hợp

Xử lý taint lịch sử trình duyệt, trước khi thông báo được gửi đến người dùng, ta sẽ tiến hành phân tích dữ liệu log được lấy từ hàng đợi của dịch vụ trong quá trình chạy. Nếu dữ liệu phù hợp với mục đích theo dõi sẽ được lưu lại và đồng thời chỉnh sửa để người dùng có thể nhận được thông báo chi tiết hơn liên quan đến truy cập thông tin nhạy cảm trên lịch sử của trình duyệt. Quá trình phân tích taint sẽ được thực hiện khi đang xử lý từng LogEntry và sau đây là các bước và sơ đồ luồng xử lý taint vi phạm:

- Bước 3.1: Kiểm tra ứng dụng có thuộc danh sách đen không
- Bước 3.2: Kiểm tra truy cập vi phạm
- Bước 3.3: Đưa vào danh sách đen

- Bước 3.4: Điều chỉnh thông báo của ứng dụng



Hình 3.4. Luồng Xử Lý Browser History Vi Phạm

3.1.4. Điều chỉnh và gửi thông báo

Dựa vào kết quả kiểm tra vi phạm truy cập theo luồng xử lý như hình 3.4 ở trên, nếu vi phạm chúng ta sẽ điều chỉnh thông báo dựa theo mức độ vi phạm. Mỗi thông báo thuộc phần cải tiến sẽ đồng thời hiển thị đèn LED theo cường độ và tần suất xác định trước theo cải tiến. Ngoài ra hệ thống cũng phát âm thanh riêng biệt theo mức độ truy xuất thông tin nhạy cảm của người dùng. Các thông báo trước khi được đưa lên giao diện của hệ thống đều được điều chỉnh phù hợp

với mức độ bảo mật bị truy cập như trên. Quá trình gửi thông báo cho người dùng sẽ được ứng dụng thực hiện như sau:

Tạo mới một đối tượng thuộc lớp Notification trong gói adroid.app. NotificationManager, lớp này thực hiện việc gửi thông báo đến người dùng. Để khởi tạo đối tượng trên cần truyền vào một đối số dưới dạng đối tượng Builder thuộc gói adroid.app.Notification.Builder. Khi khởi tạo đối tương Builder hệ thống sẽ truyền các thông tin sau qua các hàm tương ứng:

- Tiêu đề thông báo - setContentTitle("TainDroid alert")
- Tên ứng dụng – setContextText(appName)
- Biểu tượng setSmallIcon(R.drawable.icon)
- Nội dung – bigText(appName + "\n" + ipAddress + "\n" + taint)

Sau khi khởi tạo xong Builder, tiếp tục tạo ra một đối tương mới thuộc lớp Bundle trong gói android.os.Bundle. Nó đảm nhiệm việc lưu ánh xạ từ các giá trị chuỗi thành các kiểu có thể phân chia. Khi tạo đối tương sẽ đưa các giá trị tương ứng với các hằng số đã định nghĩa sẵn của thông báo vào đối tượng Bundle tương ứng các hàm sau:

putString(KEY_APPNAME, appName)
putString(KEY_IPADDRESS, ipAddress)
putString(KEY_TAINT, taint)
putString(KEY_DATA, data)
putString(KEY_ID, id)
putString(KEY_TIMESTAMP, timestamp)

Khi khởi tạo thành công Bundle, tiếp tục tạo một đối tượng thuộc lớp Intent của gói android.content. Intent là một miêu tả trừu tượng của hoạt động được thực thi, nó có thể dùng để thực hiện một hoạt động thông qua liên kết android.app.Activity để gửi các thành phần liên kết đến liên kết Context#startservice hoặc Context#bindService để tạo liên kết tầng dưới. Nó cũng cung cấp một cơ sở cho việc thực hiện kết nối thời gian thực sau này giữa mã nguồn bên trong các ứng dụng. Sử dụng chủ yếu trong việc thực hiện các hoạt động, nơi nó có thể gọi là cầu nối giữa các hoạt động. Nó chủ yếu dựa trên cấu trúc dữ liệu được gắn với một đặc tả trừu tượng của một hành động được thực thi. Khi tạo intent thì sẽ gán luôn các cờ và bundle đã tạo ở trên theo cách sau:

addFlags(Indent.FLAG_ACTIVITY_NO_HISTORY)
putExtras(extras)

Sau khi tạo xong một intent, hệ thống sẽ ra một đối tượng thuộc lớp PendingIntent trong gói android.app. Đối tượng này để gán nội dung cho một thông báo, nó được miêu tả như một intent và hành động cụ thể để thực thi. Nó có thể được tạo bởi các liên kết #getActivity, #getActivities, #getBroadcast và #getService. Các đối tượng trả lại có thể được điều khiển bằng các ứng dụng khác. Việc trao các PendingIntent cho ứng dụng khác sẽ giúp ứng dụng có quyền để thực hiện các hoạt động cụ thể như của chính ứng dụng đó, nhưng cũng nên thận trọng về việc tạo ra các PendingIntent. Ví dụ intent tạo ra phải được xác định tên riêng biệt theo thành phần cụ thể và đảm bảo nó không có khả năng gửi đi những nơi không được phép. Sau khi tạo thành công một PendingIntent, ta phải truyền nó vào cho thông báo theo cách gán lệnh: Notification.contentIndent = myPendingIntent. Tiếp theo sẽ gán các thông số hiển thị đèn LED của thông báo như sau:

- ledOnMs - Số minisecond sáng đèn led khi đang bật
- ledOffMS - Số minisecond sẽ tắt đèn led
- ledARGB - Màu của đèn led, phần cứng của thiết bị sẽ hiện thị tương đối chính xác
- Flags - Cờ mức độ ưu tiên cho thông báo, có một số loại như sau:
    - PRIORITY_MIN - Mức thấp nhất, các item có thể không được hiện cho người dùng ngoài trừ các tình huống đặt biệt như thông báo chi tiết.
    - PRIORITY_LOW - Thông báo không quan trọng, mức ưu tiên thấp. Giao diện có thể chọn hiển thị nhỏ hơn thông thường hay ở một vị trí khác trong danh sách so với các thông báo khác.
    - PRIORITY_DEFAULT - Nếu ứng dụng không có quyền riêng thì sẽ dùng giống như các thông báo thông thường.
    - PRIORITY_HIGH - Mức ưu tiên cao cho các thông báo ở mức quan trong hơn. Giao diện có thể đưa ra cách hiển thị rộng hơn hoặc ở các vị trí khác nhau.
    - PRIORITY_MAX - Mức ưu tiên cao nhất cho ứng dụng, thông báo sẽ được yêu cầu hiện thị như thông báo hệ thống.

Cuối cùng sẽ tạo một đối tượng thuộc lớp NotificationManager của gói android.app để thực hiện việc gửi thông báo. Đối tượng này sẽ có nhiệm vụ gửi thông báo tới người dùng khi các sự kiện xảy ra. Nó hỗ trợ ứng dụng đưa đến người dùng những gì xảy ra từ tầng dưới của hệ thống, các thông báo có nhiều dạng khác nhau như sau:

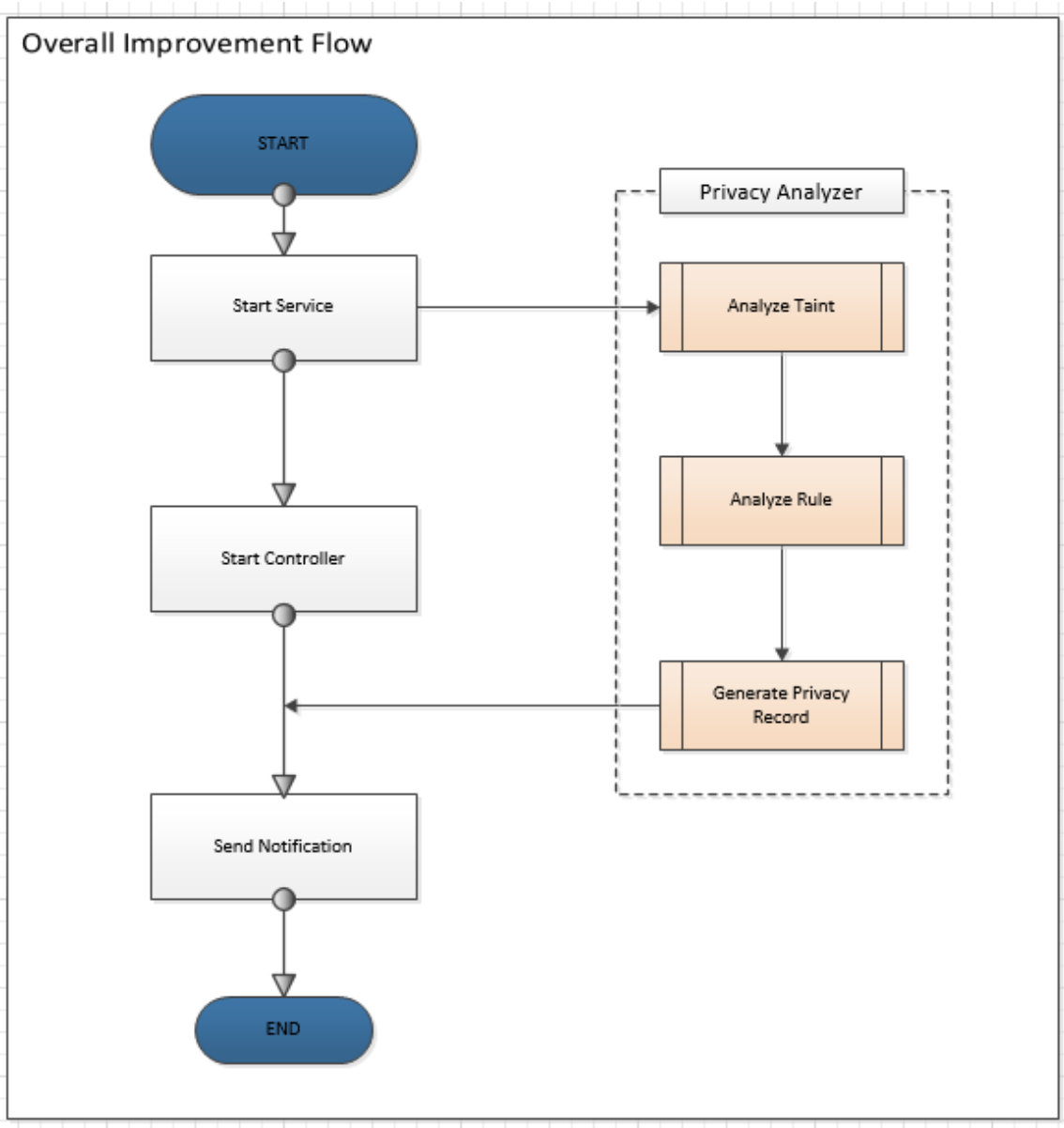
- Một biểu tượng trên thanh bar và có thể truy xuất qua các launcher khi người dùng chọn nó.
- Bật hay nhấp nháy đèn LED trên thiết bị.
- Cảnh báo người dùng bằng nhấp nháy đèn flash của camera hay phát nhạc, rung thiết bị.

3.2. Giải pháp cải tiến tổng quát

Trên cơ sở cải tiến kiểm soát truy cập thông tin nhạy cảm của lịch sử trình duyệt ở trên, tôi xin được đề xuất giải pháp cải tiến tổng quát như sau. Chúng ta sẽ xây dựng một bộ phân tích chính sách (Privacy Analyzer), nó sẽ phân tích các taint theo các luật của chính sách (Privacy rule) đầu vào xác định và trả lại kết quả dữ liệu taint có vi phạm chính sách can thiệp hay không. Khi dịch vụ của TaintDroid được chạy, các module đăng ký dịch vụ có thể nhận được các taint. Module có thể sử dụng bộ phân tích chính sách này để kiểm tra vị phạm chính sách của một taint bất kỳ.

Bộ phân tích chính sách là module xử lý tất cả các loại taint được định nghĩa sẵn trong TaintDroid. Các luật của bộ xử lý sẽ được ánh xạ tương ứng đến các loại taint. Khi xử lý một taint nó sẽ phân tích quyền truy xuất đến vùng thông tin của ứng dụng tương ứng. Sau đó sẽ kiểm tra các quyền hiện hữu của ứng dụng so với các luật đầu vào để xác định taint có vi phạm chính sách truy cập hay không. Như vậy các module khác có thể tùy biến các luật đầu vào để kiểm tra vi phạm của một taint và đưa ra các xử lý cho riêng mình. Ví dụ một module có thể đưa ra 2 luật cùng lúc, luật 1 có key là vị trí và một luật có key là camera. Và yêu cầu bộ phân tích chính sách kiểm tra taint có vi phạm truy xuất dữ liệu tương ứng không?

Căn cứ vào hệ thống thực tế và nhu cầu thiết kế một giải pháp tổng thể hỗ trợ các module có thể kiểm tra taint thời gian thực. Việc xử lý dữ liệu phải được gắn kèm với dịch vụ của hệ thống, luồng xử lý của giải pháp cải tiến sẽ được thực hiện như miêu tả trong hình 3.5 dưới đây.



Hình 3.5. Luồng Cải Tiến Tổng Quát

Bộ phân tích chính sách sẽ phân tích dữ liệu của taint đầu vào, phân tích xem có vi phạm chính sách can thiệp thông tin theo yêu cầu của luật đầu vào. Và cuối cùng sẽ trả về kết quả dưới dạng bản ghi chính sách can thiệp (Privacy record). Nó sẽ chứa các logic riêng, kiểm tra quyền truy cập của mỗi ứng dụng so với luật được yêu cầu. Bộ xử lý có thể xử lý nhiều luật cùng lúc và kết hợp linh hoạt các luật từ đó đưa ra kết quả dưới dạng bản ghi chính sách.

Ví dụ áp dụng: Module giám sát truy cập thông tin nhạy cảm trình duyệt đã nêu ở trên, nó có thể yêu cầu phân tích chính sách với một luật BrowserHistory (ví dụ 1 luật như sau: ruleName = HistoryBrowser, và 3 Rule keywords: = UserName, Password, CreditCard). Sau đó sẽ nhận được bản ghi chính sách can thiệp cho taint yêu cầu. Từ đó module có thể phân tích kết quả trả về và xử lý

logic riêng của mình. Khi đó nó có thể quyết định điều chỉnh thông báo và gửi cho người dùng hay không? Việc phân tích chính sách can thiệp gồm 4 bước: Phân tích taint (Analyze taint), Phân tích luật (Analyze rule), tạo bản ghi chính sách can thiệp (Generate privacy record) và sau đây là chi tiết từng bước.

3.2.1. Phân tích taint

Khi dịch vụ TaintDroidNotifyService được chạy, các module đăng ký dịch vụ có thể nhận được các taint đang được ứng dụng truy cập và gửi đi theo luồng. Module sẽ chuẩn bị các luật của chính sách, gửi kèm thông tin về taint để xử lý.

Dịch vụ thông báo TaintDroid được mở rộng từ dịch vụ Android, nên nó sẽ lắng nghe tất các dữ liệu trong hàng đợi của HĐH. Dữ liệu trong hàng đợi đều được định nghĩa theo cấu trúc theo lớp hàng đợi của Android. Mỗi khi có khối hàng đợi được truyền đi, TaintDroid sẽ lấy dữ liệu bằng phương thức take() của lớp Queue. Như vậy mỗi hàng đợi block điều được TaintDroid xử lý và khóa theo taint đã định nghĩa để thông báo cho người dùng khi cần thiết. Dưới đây là đoạn mã chương trình minh họa xử lý trên.

```
private volatile boolean doRead = false;
private Thread readThread = null;
private class Consumer implements Runnable {
    private final BlockingQueue queue;
    Consumer(BlockingQueue q) { queue = q; }
    public void run()
    {
        LogEntry prev = null;
        while(doRead) {
            try {
                LogEntry le = (LogEntry)queue.take();
                processLogEntry(le);
            }
            catch (InterruptedException e) {
                Log.e(TAG, "Could not read log entry: " + e.getMessage());
            }
        }
    }
}
```

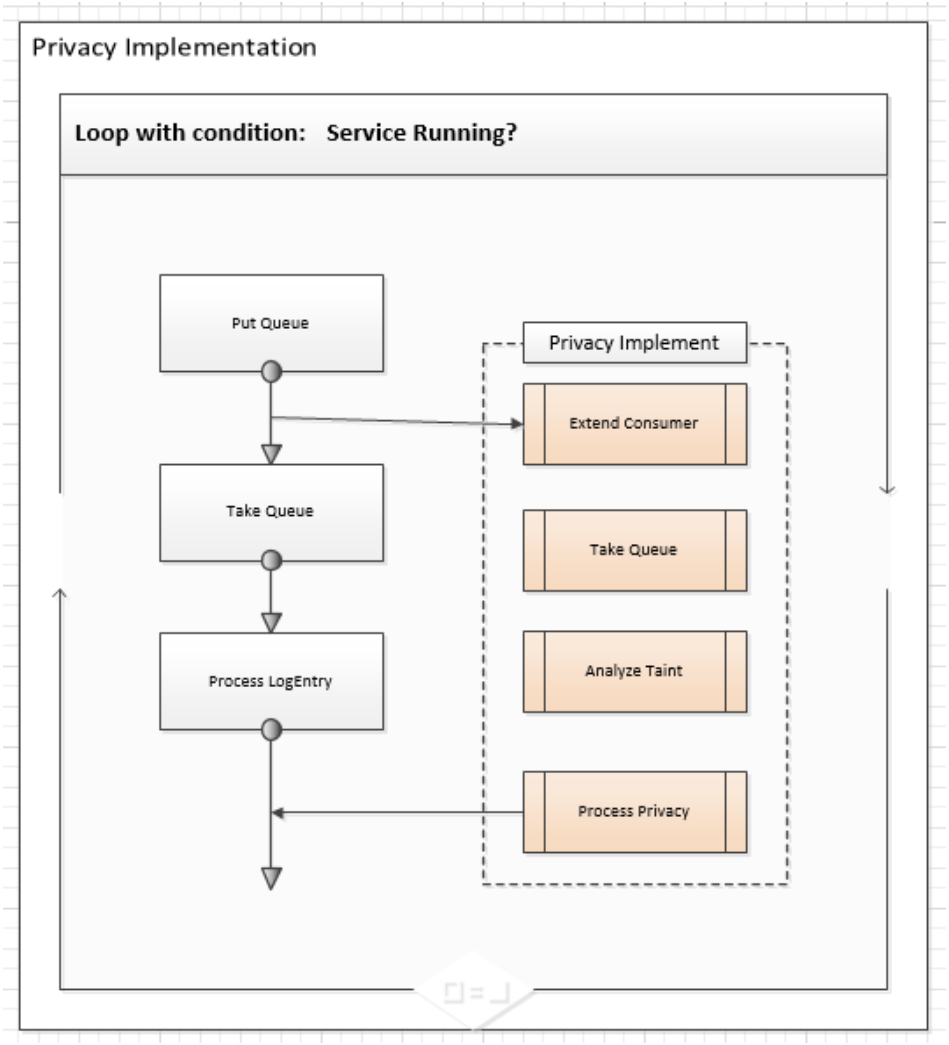
Với mỗi hàng đợi nhận được từ HĐH, TaintDroid sẽ chuyển đổi sang thực thể LogEntry định nghĩa riêng để xử lý. Hệ thống sẽ trích xuất ra thông điệp, thời gian hay id tiến trình từ hàng đợi. Sau đó sẽ lọc ra các thông tin cụ thể từ thông điệp như : Ip, Taint, Data, … và dưới đây là hình ảnh minh họa một phần phương thức xử lý LogEntry.

```
private void processLogEntry(LogEntry le) {
    String timestamp = le.getTimestamp();
    String msg = le.getMessage();
    boolean taintedSend = isTaintedSend(msg);
    boolean taintedSSLSend = isTaintedSSLSend(msg);
    if(taintedSend || taintedSSLSend) {
        String ip = get_ipaddress(msg);
        String taint = get_taint(msg);
        String app = get_processname(le.getPid());
        String data = get_data(msg);
        if (taintedSSLSend)
            ip=ip+" (SSL)";
```

Như vậy mỗi module mới cần xử lý taint cho riêng mình thì cần phải mở rộng lớp Consumer của dịch vụ TaintDroidNotifyService và lấy taint để xử lý riêng cho module đó. Module mới có thể lấy được thông tin taint vì lớp Consumer thực thi giao diện Runnable của gói dịch vụ Android. Để thực hiện việc này ta cần sửa lại lớp Consumer để hỗ trợ các module khác có thể mở rộng lớp trên như hình minh họa sau:

```
public class Consumer implements Runnable {
    private final BlockingQueue queue;
    Consumer(BlockingQueue q) { queue = q; }
    public void run()
    {
        LogEntry prev = null;
        while(doRead) {
            try {
                LogEntry le = (LogEntry)queue.take();
                processLogEntry(le);
            }
            catch (InterruptedException e) {
                Log.e(TAG, "Could not read log entry: " + e.getMessage());
            }
        }
```

Về tổng thể một module mới khi được gắn vào dịch vụ của TaintDroid thì phải mở rộng nó, thực thi logic riêng của mình và dưới đây là hình ảnh minh họa luồng xử lý cho mỗi module mới.



Hình 3.6. Luồng xử lý taint của module thực thi

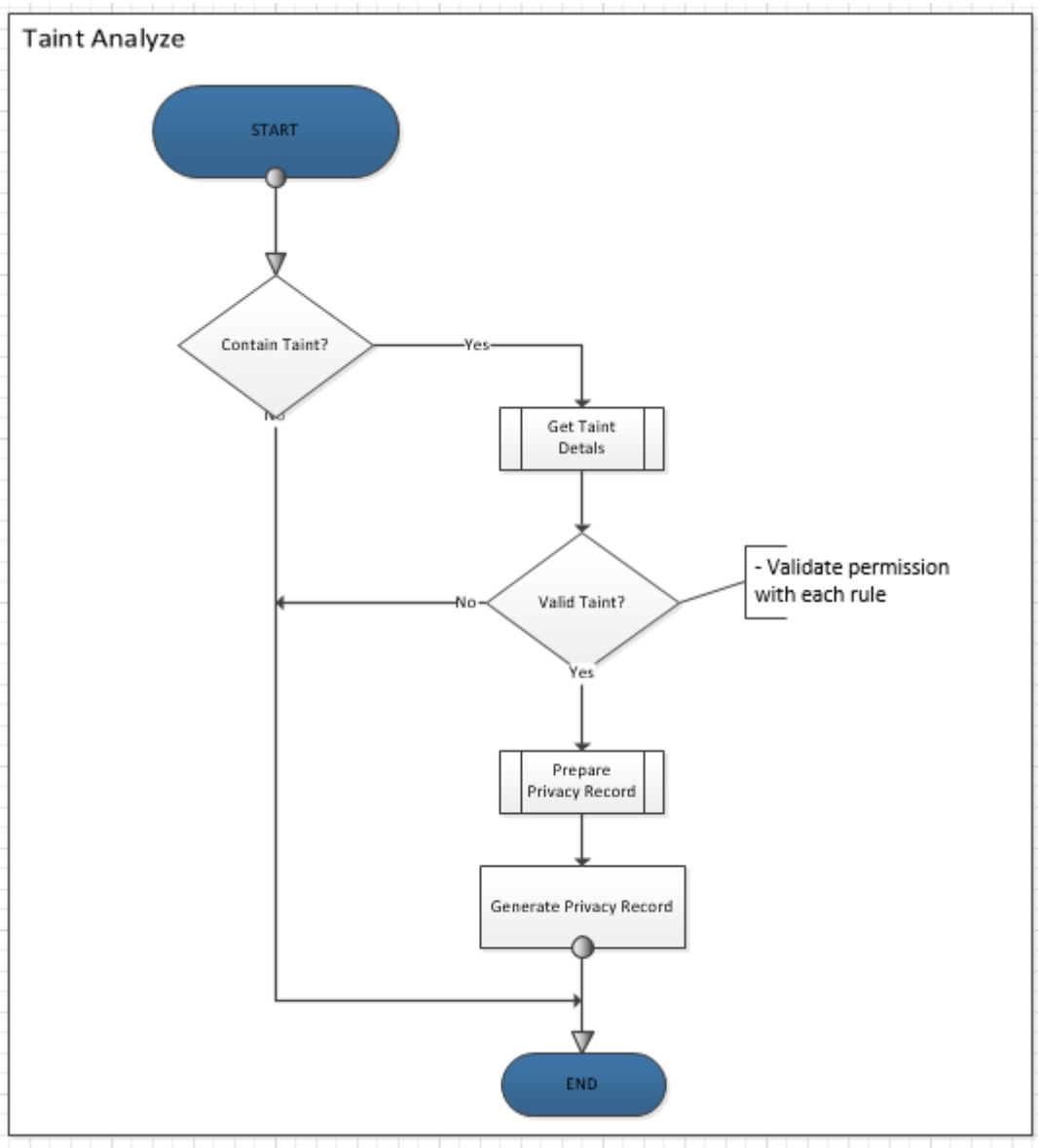
3.2.2. Phân tích luật

Khi một module mới được mở rộng từ dịch vụ của TaintDroid, nó sẽ nhận được các hàng đợi của dịch vụ và chuẩn bị luật của riêng mình. Sau đó nó sẽ gọi thư viện của PrivacyAnalyzer để xử lý và kiểm tra vi phạm chính sách. Thông thường các module tao mới để kiểm tra truy cập thông tin nhạy cảm, nên nó sẽ cung cấp các bộ luật tương ứng đến thông tin nhạy cảm. Để chuẩn bị thông tin đầu vào trước khi xử lý, cần phải xác định danh sách các luật để kiểm tra. Và dưới đây là cấu trúc của một luật, chính là một thực thể tương ứng của lớp PrivacyRule:

- RuleName: Là hằng số được định nghĩa sẵn theo enum có tên EnumRuleName, các hằng này tương ứng với các loại taint mà PrivacyAnalyzer sẽ xử lý, chúng ta có thể mở rộng nếu TaintDroid có thay đổi.
- RuleKeys: Danh sách các key thuộc luật, dữ liệu kiểu mảng ký tự. Nó chứa danh sách các ký tự cần kiểm tra tương ứng với luật. Ví dụ: {“username”, “password”, “email”}.

- EnumRuleName: Location, AddressBoox, Microphone, Phone_Number, GPS_Location, NETBased_Location, Last_Known_Location, Camera, Accelerometer, SMS, IMEI, IMSI, ICCID, Device_Sserial_Number, User_Account_Information, Browser_History.

Tùy thuộc vào các luật truyền vào, các logic riêng sẽ được thực hiện để kiểm tra vi phạm truy cập và hình 3.7 dưới đây là sơ đồ minh họa luồng phân tích.



Hình 3.7. Luồng phân tích taint

Để phân tích chính sách như trên cần xây dựng một lớp trong hệ thống hiện có PrivacyAnalyzer. Lớp có một phương thức được công khai cho các module khác sử dụng Analyze. Ngoài ra lớp còn chứa các logic riêng để phân tích và đều được tạo dưới dạng biến hay phương thức riêng. Đoạn mã dưới đây thể hiện cấu trúc rút gọn của lớp PrivacyAnalyzer:

```
import java.util.ArrayList;

/**
 * Analyze the taint data is matched to rules
 * @author ChungNN
 */
public class PrivacyAnalyzer {
    private static PrivacyAnalyzer instance = new PrivacyAnalyzer();
    private static Hashtable<Integer, String> ttable = new Hashtable<Integer, String>();
    static {

     * Get the instance of class
    public static PrivacyAnalyzer getInstance() {

     * Analyze privacy from the queue message is matched with rules
    public PrivacyRecord Analyze(String msg, ArrayList<PrivacyRule> rules)

     * Analyze privacy from the queue message is matched with rules
    private PrivacyRecord AnalyzeTaint(String msg, ArrayList<PrivacyRule> rules)

     * Process a rule is matched with queue message
    private boolean IsValideRule(PrivacyRule rule, String msg)

     * Get the ip address from queue message
    private String get_ipaddress(String msg) {

     * Get taint from taint data
```

3.2.3. Tạo bản ghi chính sách can thiệp

Khi việc phân tích các luật trên taint đầu vào hoàn thành, bộ phân tích chính sách sẽ chuẩn bị dữ liệu để tạo ra bản ghi chính sách và trả lại cho module gọi. Bản ghi sẽ miêu tả chi tiết tình trạng vi phạm chính sách từ trạng thái đến đặc tả chi tiết về vi phạm. Để tạo tra cấu trúc bản ghi cần tạo ra một lớp mới có tên PrivacyRecord và phải công khai để các module khác có thể truy xuất và dưới đây là hình ảnh minh họa cấu trúc của lớp này:

```
package org.appanalysis;

/**
 * The privacy record describes detail of violation.
 * @author ChungNN
 */
public class PrivacyRecord {
    private boolean isViolated = false;
    private ViolatedLevel violatedLevel = ViolatedLevel.Low;
    private String description;

    public boolean isViolated() {

    public void setViolated(boolean isViolated) {

    public String getDescription() {

    public void setDescription(String description) {

    public ViolatedLevel getViolatedLevel() {

    public void setViolatedLevel(ViolatedLevel violatedLevel) {

    public enum ViolatedLevel {
        Low, Medium, High;
```

Một bản ghi chính sách thể hiện được kết quả có vi phạm hay không và các thông tin khác được miêu tả dưới đây:

- isViolated: Xác định có vi phạm hay không?
- Description: Miêu tả chi tiết về vi phạm
- ViolatedLevel: Mức độ vi phạm từ thấp đến mới cao

Ví dụ minh họa dưới đây về việc tạo mới một module để mở rộng lớp vừa được sửa đổi. Lớp mới sẽ nhận các hàng đợi từ hệ thống và gọi phương thức Analyze() để kiểm tra vi phạm chính sách. Lớp truyền vào dữ liệu của hàng đợi và một luật AddressBox rồi kiểm tra 2 từ khóa "ChungNN" và "Email". Sau khi xử lý xong, bộ phân tích sẽ trả lại bản ghi chính sách cho biết taint này có vi phạm truy cập danh bạ và lấy tên danh bạ = "ChungNN" hay địa chỉ email không?

```
/**
 * A new module implement processing taint by using privacy analyzer.
 * @author ChungNN 2017/04/15
 */
public class NewModule extends TaintDroidNotifyService.Consumer {
    NewModule(TaintDroidNotifyService taintDroidNotifyService, BlockingQueue<?> q) {
        taintDroidNotifyService.super(q);
        // TODO Auto-generated constructor stub
        LogEntry le;
        try {
            le = (LogEntry)q.take();
            ArrayList<PrivacyRule> rules = new ArrayList<PrivacyRule>();

            //Create rule 1
            String[] keys = {"ChungNN", "Email"};
            PrivacyRule rule1 = new PrivacyRule(EnumRuleName.AddressBoox, keys);
            rules.add(rule1); //Create rule 1

            //Create rule 2, 3, ...

            //Analyze taint with matched rules
            PrivacyRecord record = PrivacyAnalyzer.getInstance().Analyze(le.getMessage(), rules);
            if(record.isViolated()) {
                // TODO implement own logic of module...
            }
            else {
                // TODO implement own logic of module...
```

3.3. Các vấn đề trong quá trình cải tiến

3.3.1. Các lưu ý trong quá trình cài đặt

Trong quá trình thực tế xây dựng hệ thống TaintDroid trên máy ảo chạy HĐH Ubuntu 14, có một số vấn đề cần lưu ý để có thể xây dựng thành công hệ thống. Trước khi bắt tay vào công việc xây dựng cần tham khảo các tài liệu về xây dựng hệ thống Android từ các nguồn uy tín, đặc biệt từ web site phát triển Android[1]. Webstie này đồng thời cũng hướng dẫn chi tiết cách cấu hình HĐH

[1] Website phát triển Android của Google có địa chỉ tại https://developer.android.com, hỗ trợ lập trình viên nghiên cứu cũng như phát triển các ứng dụng chạy HĐH Android.

Unbuntu để có thể phát triển ứng dụng Android. TaintDroid là một hệ thống được sửa đổi từ HĐH Android, nên tất cả các mã nguồn của HĐH gốc vẫn được giữ nguyên mà chỉ nhúng thêm module riêng. Vậy nên việc xây dựng TaintDroid đều cần thực hiện các bước cấu hình như khuyên dùng của một HĐH Android thông thường.

Trước khi lấy mã nguồn TaintDroid cần phải đảm bảo HĐH đã được cập nhật mới nhất cùng các gói công cụ. Khi thực hiện biên dịch hệ thống cũng phải đảm bảo cấu hình HĐH hỗ trợ ảo hóa. Sau khi sửa đổi bất kỳ module nào nên kiểm tra ngay trên môi trường phát triển, tránh biên dịch toàn bộ hệ thống sẽ mất nhiều thời gian. Khi ghi (flash) các tệp ảnh lên thiết bị cần tuân thủ các bước từ website android source [3] và lưu ý hạn chế tối đa số lần ghi để tránh hư hỏng thiết bị do ghi nhiều lần.

Sau khi đã cài đặt thành công TaintDroid lên thiết bị, có một số ứng dụng quan trọng chưa hoạt động hay chưa có trong hệ thống như Google play service. Cần tìm các tệp ứng dụng từ các nguồn tin cậy như Google, Amazon, … để cài đặt, tránh gây lỗi đến ứng dụng TaintDroid notify hay toàn bộ hệ thống. Trong quá trình xây dựng hệ thống cũng có thể gặp một số lỗi kiến không thể biên dịch được và dưới đây là một số lỗi thường gặp.

3.3.2. Các lỗi thường gặp khi xây dựng hệ thống

- Exited sync due to fetch errors:

▪ Nguyên nhân: Khi chạy lệnh "repo sync" để đồng bộ mã nguồn TaintDroid về máy tính, lỗi xuất hiện thông thường do nguyên nhân có một lần chạy từ trước và không thành công. Hay việc gỡ bỏ mã nguồn đang động bộ chưa thành công hoặc có cấu hình chưa được xỏa bỏ.

▪ Khắc phục: Thực hiện lệnh “repo sync --force-sync” để yêu cầu đồng bộ ghi đè mã nguồn và các cấu hình.

- Flex: command not found:

▪ Nguyên nhân: Khi tiến hành thực hiện lệnh “make –j4” để tạo các tệp ảnh của hệ thống. Lỗi xảy ra khi hệ thống chưa cài đặt đúng hay chưa cập nhật gói flex.

▪ Khắc phục: Chạy 2 lệnh sau để đảm bảo cập nhật hệ thống và cài đặt gói flex:

- sudo apt-get update
- sudo apt-get install flex

- Project dalvik not found in Taintdroid:

▪ Nguyên nhân: Khi thực hiện việc lấy mã nguồn bằng lệnh “repo sync”, một số gói cho máy ảo dalvik không được tải về trước để thực hiện lấy mã nguồn.

- Khắc phục: Cần thực hiện việc tải toàn bộ các gói liên quan bằng câu lệnh sau:

repo forall dalvik libcore frameworks/base frameworks/native frameworks/opt/telephony system/vold system/core device/samsung/manta device/samsung/tuna \packages/apps/TaintDroidNotify -c 'git checkout -b taintdroid-4.3_r1 --track github/taintdroid-4.3_r1 && git pull'

Một số lưu ý: Không thực hiện lệnh "make clean" khi không có bất khì thay đổi nào của hệ thống, lệnh này sẽ xóa các dữ liệu của bản biên dịch trước. Hệ thống TaintDroid có dự liệu lớn khoảng 1GB, nên nếu xóa thì khi tạo lại sẽ mất nhiều thời gian (với cấu hình máy tính trong thực nghiệm phải mất 3 giờ để thực hiện).

3.4. Đóng góp trong cải tiến

Trong quá trình thực hiện luận văn hệ thống đã được thay đổi để hoạt động linh hoạt và có tính ổn định cao hơn. Các thay đổi không làm ảnh hưởng đến các luồng xử lý cũng như dữ liệu của hệ thống sẵn có. Thay đổi chỉ nhằm mục đích cải tiến hệ thống hiện có cũng như cung cấp các khả năng mở rộng cho các module khác viết thêm vào hệ thống. Để hạn chế chiếm dụng nhiều bộ nhớ của hệ thống, đảm bảo hiệu năng của hệ thống vẫn tốt cần hạn chế số lượng hàng đợi xử lý trong một tiến trình. Vì vậy cần xác định giá trị tối đa của mảng LogEntry trong mỗi một tiến trình xử lý.

```
private BlockingQueue logQueue;
private static final int LOGQUEUE_MAXSIZE = 4096;

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    if(isRunning) {
        return START_NOT_STICKY;
    }

    logQueue = new ArrayBlockingQueue<LogEntry>(LOGQUEUE_MAXSIZE);
    this.captureThread = new Thread(new Producer(logQueue));
    captureThread.setDaemon(true);
```

Để hỗ trợ các module khác có thể truy xuất trực tiếp khi dịch vụ TaintDroid, cần công khai lớp Consumer vì lớp này thực thi giao diện Runnable được kế thừa trực tiếp từ thư viện Service của Android. Việc này đồng thời cũng giúp hệ thống được dễ dàng mở rộng linh hoạt hơn.

```
public class Consumer implements Runnable {
    private final BlockingQueue queue;
    Consumer(BlockingQueue q) { queue = q; }
    public void run()
    {
        LogEntry prev = null;
        while(doRead) {
            try {
                LogEntry le = (LogEntry)queue.take();
                processLogEntry(le);
            }
            catch (InterruptedException e) {
                Log.e(TAG, "Could not read log entry: " + e.getMessage());
            }
        }
```

Chương 4. Kết quả thử nghiệm

4.1. Môi trường thử nghiệm

TaintDroid được xây dựng từ HĐH Android, nên nó cũng yêu cầu phải có môi trường để phát triển và thử nghiệm tương ứng. Phần cứng cũng như phần mềm đều phải tương thích với HĐH của phiên bản TaintDroid, trong khuôn khổ luận văn việc cải tiến được thực hiện trên TaintDroid dành cho Android 4.3. Trong quá trình tìm hiểu các chỉ dẫn từ website của TaintDroid cũng như website phát triển Android của Google [2], để có môi trường thử nghiệm tốt cần phải tuân thủ các yêu cầu từ phần cứng đến phần mềm. Dưới đây là môi trường đã được thử nghiệm trong khuôn khổ luận văn:

- Thiết bị: Laptop: HP Probook, CPU: Core i5 (4 cores, 4 processors), RAM: 8GB, HDD: 1TB hỗ trợ công nghệ ảo hóa.
- Phần mềm: HĐH: Windows 10 Enterprise (English) x64bit, VMware Workstation 12.0 Pro. Và máy ảo VMWare cũng phải có cấu hình khuyến cáo như sau:
  - Phần cứng:
    - CPU: 2 cores, 2 processors
    - RAM: 4GB
    - HDD: 100GB.
  - Phần mềm:
    - HĐH: Ubuntu 14.04
    - Eclipse MARS 2 for Unbuntu

4.2. Thiết bị thử nghiệm

- Điện thoại di động: Google Nexus 4
- Hệ điều hành: Android OS, v4.3 (Jelly Bean)
- CPU: Quad-core 1.5 GHz Krait
- Chipset: Qualcomm APQ8064 Snapdragon S4 Pro
- RAM: 2GB
- GPS: With A-GPS, GLONASS

4.3. Chạy ứng dụng

Để chạy thử các cải tiến trên thiết bị thật, chúng ta cần phải thực hiện các bước sau.

- Bước 1 - Biên dịch hệ thống:
  - Bật máy ảo Ubuntu
  - Chạy Eclipse

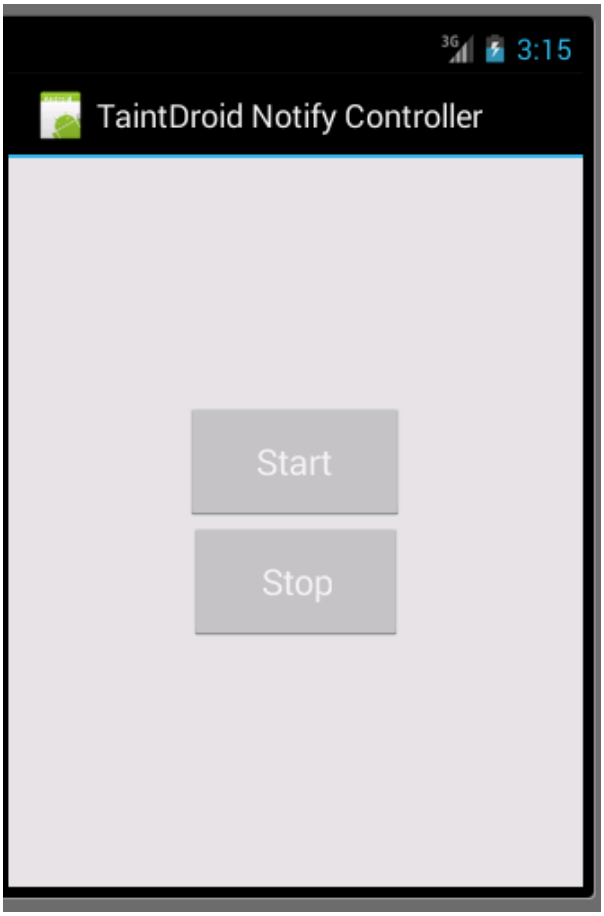
- Biên dịch ứng dụng TaintDroidNotify
- Mở command line
- Biên dịch toàn bộ hệ thống TaintDroid

▪ Bước 2 - Cài đặt TaintDroid lên thiết bị:

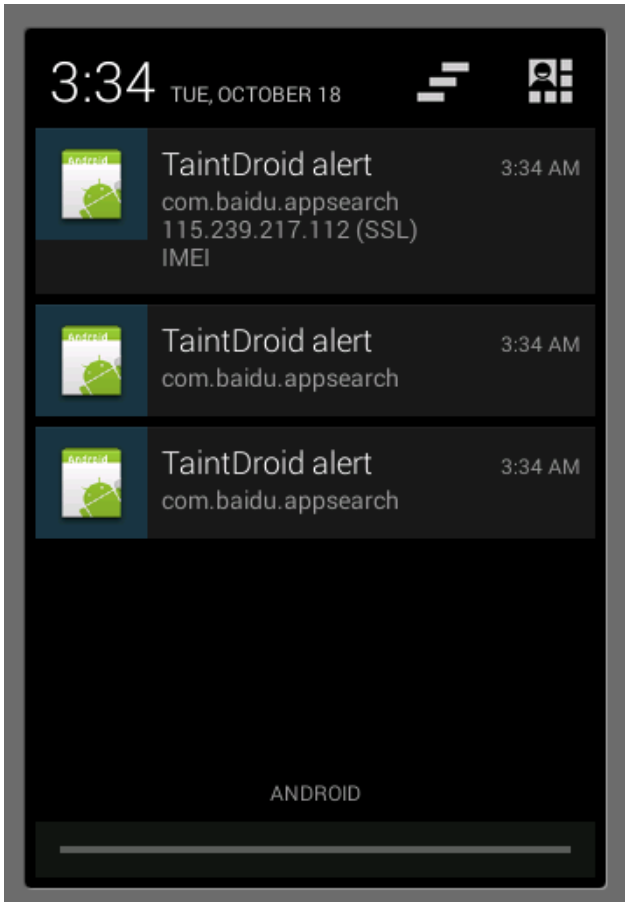
- Kết nối điện thoại với máy tính
- Cài đặt USB driver cho Windows
- Khởi động điện thoại về chế độ BootLoader
- Bật máy ảo Ubuntu
- Mở command line
- Ghi các tệp ảnh sau vào thiết bị bằng các lệnh tương ứng:

```
$ fastboot flash boot boot.img
$ fastboot flash system system.img
$ fastboot flash userdata userdata.img
```

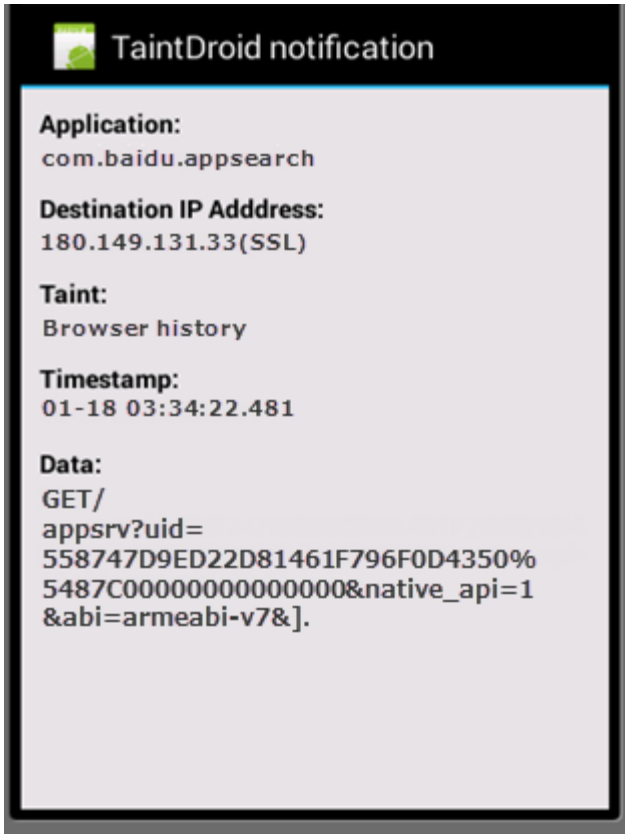
▪ Bước 3 - Chạy ứng dụng: Chọn ứng dụng TaintDroid Notify từ màn hình chính, chọn nút "Start" để bắt đầu theo dõi và dưới đây là một số hình ảnh minh hoạt hoạt động kiểm tra của cải tiến truy cập lịch sử trình duyệt.



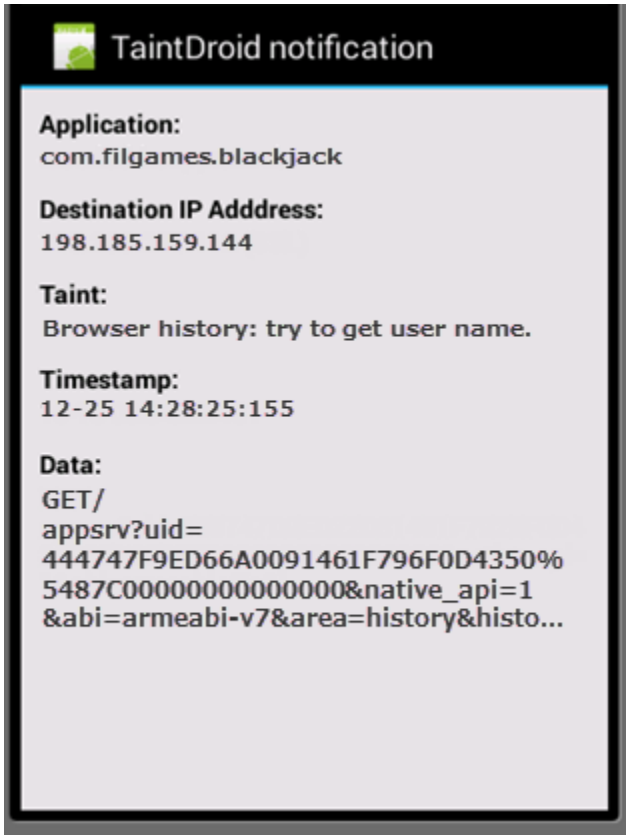
Hình 4.1: Giao diện chương trình thực thi



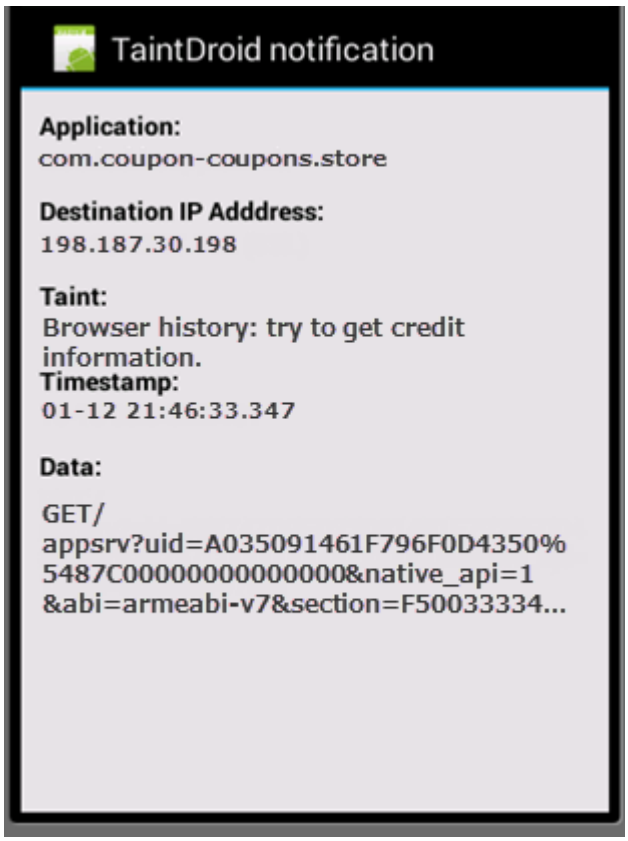
Hình 4.2: Danh sách thông báo



Hình 4.3. Thông báo truy xuất lịch sử trình duyệt



Hình 4.4. Thông báo truy xuất tên đăng nhập



Hình 4.5. Thông báo truy xuất mã số thẻ tín dụng

4.4. Đánh giá cải tiến

Hệ thống sau khi cải tiến (TaintDroid') đã hoạt động bình thường và không có bất kỳ thay đổi nào về mặt kiến trúc cũng như luồng xử lý ảnh hưởng đến hệ thống cũ. Và dưới đây là một số tiêu chí đánh giá nổi bật so với HĐH gốc (Android) và hệ thống TaintDroid trước khi cải tiến.

4.4.1. MacrobenchMarks

Mỗi thực nghiệm tiến hành đo 50 lần và quan sát khoảng 95% thời lượng. Mỗi trường hợp đều bỏ qua lần chạy đầu để không tính thời gian khởi tạo và kết quả được ghi nhận trong bảng 4.1.

Bảng 4.1: Kết quả MacrobenchMarks (1.000 thông điệp)

	Android	TaintDroid	TaintDroid'
Tải ứng dụng	63 ms	65 ms	68 ms
Tạo danh bạ	348 ms	367 ms	372 ms
Đọc danh bạ	101 ms	119ms	125 ms
Gọi điện	96 ms	106 ms	112 ms
Chụp ảnh	1718 ms	2216 ms	2247 ms

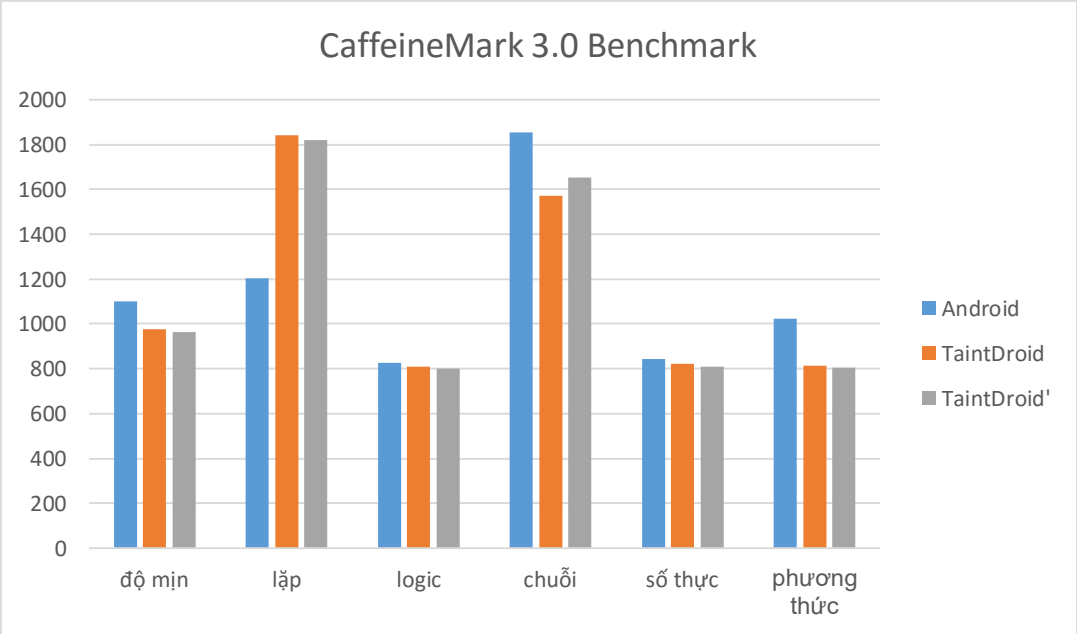
- Thời gian tải ứng dụng: Thời gian tải ứng dụng được đo khi bộ quản lý hoạt động của Android (Android's Activity Manager) nhận một lệnh và bắt đầu một thành phần hoạt động tới khi luồng hoạt động được hiển thị. Thời gian này bao gồm việc phân giải ứng dụng bởi Activity Manager, IPC và hiển thị đồ họa. TaintDroid' chỉ thêm gần 3% chi phí các hoạt động đồ họa dùng các thư viện đồ họa nguyên gốc.
- Địa chỉ liên hệ: Một ứng dụng được hiệu chỉnh để tạo, đọc và xóa toàn bộ địa chỉ liên lạc của điện thoại, với bài kiểm tra cả đọc và ghi. Tạo ra 3 giao dịch SQL với việc đọc chiếm 2 giao dịch. TaintDroid' tăng thêm 5.7% và 19% chi phí cho việc tạo và đọc toàn bộ danh bạ. Dữ liệu chưa được đánh dấu taint trước khi tạo, vì thế không cần truyền bất cứ tệp nào. Lưu ý trải nghiệm thông thường của người dùng là thời gian tạo hay xem một liên hệ chỉ mất dưới 20ms.
- Gọi điện: Chỉ số benchmark cho cuộc gọi được đo từ thời gian chạm vào biểu tượng nút "gọi điện" tới thời điểm thiết bị âm thanh ghi nhận chế độ nhận

cuộc gọi. TaintDroid' chỉ mất thêm 12ms cho mỗi lần thực hiện cuộc gọi (TaintDroid là 10ms, tương đương 10% chi phí hệ thống).

- Chụp ảnh: Chỉ số benchmark cho chụp ảnh được đo từ thời điểm người dùng chạm vào biểu tượng "chụp ảnh" đến khi hình ảnh xem trước được hiển thị. Việc đo này bao gồm thời gian chụp ảnh từ máy ảnh và lưu thành tệp vào thẻ nhớ và TaintDroid' mất 502ms (TaintDroid là 498ms). Lưu ý một vài chi phí có thể giảm được bởi việc loại bỏ truyền taint dư thừa, nghĩa là chỉ có các thẻ taint cho lần ghi dữ liệu đầu đầu tiên vào tệp là cần thiết.

4.4.2. Java MacrobenchMark

Hình 4.6 cho thấy kết quả thời gian thực thi của một chỉ số microbenchmark Java. Một cổng Android của CaffeineMark tiêu chuẩn được sử dụng. CaffeineMark chỉ dùng một số liệu lưu trữ hữu dùng cho các so sánh liên quan.



Hình 4.6. Microbenmark của overhead Java

Các kết quả đều nhất quán với việc thực thi xác định. Chi phí phát sinh bởi TaintDroid' nhỏ hơn so với các chỉ số nổi bật bởi thực hiện số học và logic. Việc truyền taint cho các hoạt động là đơn giản, bao gồm cả việc sao chép bộ nhớ nội tại phân tán., chỉ số "chuỗi" chiếm chi phí cao nhất.

Kết luận là các kết quả chỉ ra các chỉ số độc lập và lưu trữ chéo. Tài liệu CaffeineMark công bố các điểm được đáp ứng tới số lệnh Java được thực thi mỗi giây. Hệ thống Android với điểm trung bình 1121, TaintDroid là 967 và TaintDroid' là 1012. TaintDroid' chiếm chi phí 15% so với hệ thống trước khi cải tiến. Chi phí bộ nhớ cũng được đo trong lúc tính chỉ số CaffeineMark. Chỉ số chi phí là 22.5 MB trong khi hệ thống TaintDroid là 21.28 MB và Android là 22.21 MB, vậy chi phí là 4.5%. Lưu lý rằng hầu hết bộ nhớ xử lý Android được

dùng bởi môi trường chạy. Các trang nhớ thư viện nguyên gốc được chia sẻ giữa các ứng dụng để giảm tải bộ nhớ hệ thống và yêu cầu kiểm tra taint. Đưa ra điểm kiểm tra của TaintDroid' là 33 còn TaintDroid là 32 cho các biến trong môi trường thông dịch 32bit.

4.4.3. IPC MacrobenchMark

Chỉ số IPC xem xét chi phí trong khi điều chỉnh các gói. Trong thực nghiệm này các ứng dụng client-service được tạo ra để thực hiện các giao dịch nhanh nhất có thể. Dịch vụ cung cấp các đối tượng tài khoản (tên đăng nhập, số dư) và 2 giao diện setAccount() và getAccount(), thực nghiệm đo được thời gian client yêu cầu thực hiện mỗi giao diện 5.000 lần. Bảng 4.2 tóm tắt kết quả chỉ số IPC. TaintDroid' chậm hơn 29%, TaintDroid chậm hơn 27% so với Android. TaintDroid' chỉ thêm 4 byte vào mỗi đối tượng IPC, dường như nguyên nhân tăng chi phí là việc sao chép các thẻ taint liên tiếp. Cuối cùng, TaintDroid' sử dụng thêm 3% bộ nhớ so với TaintDroid, nó có thể được so sánh chi phí với chỉ số CaffeineMark.

Bảng 4.2. Kết quả kiểm tra thông lượng IPC (5.000 thông điệp)

	Android	TaintDroid	TaintDroid'
Thời gian (s)	9.24	10.03	10.15
Bộ nhớ (ứng dụng)	21.05 MB	21.76 MB	22.04 MB
Bộ nhớ (dịch vụ)	18.52 MB	19.42 MB	20.72 MB

4.5. Thảo luận

Theo kinh nghiệm thực tiễn trong quá trình nghiên cứu, phân tích và cải tiến hệ thống TaintDroid tôi nhận thấy có một số điểm cần lưu ý sau:

- Chuẩn bị dựng hệ thống: Cần tìm hiểu các kiến thức cơ bản về phát triển và xây dựng hệ thống chạy trên nền tảng Android. Từ việc cài đặt hệ thống đến việc lấy mã nguồn cũng như biên dịch hệ thống. Việc biên dịch hệ thống cũng cần một hệ điều hành ổn định, cài đặt và cập nhật đầy đủ các gói công cụ để việc xây dựng hệ thống không gặp nhiều lỗi.
- Cải tiến TaintDroid: Trước khi cải tiến hệ thống TaintDroid cần hiểu rõ cấu trúc, luồng xử lý của hệ thống hiện tại. Hiểu rõ về cấu trúc chương trình thực thi, các luồng xử lý cũng như các lớp của nó.

- Cập nhật phần mềm: Trước tiên cần lưu ý, việc cài đặt hệ thống TaintDroid đòi hỏi phải ghi toàn bộ hệ thống và việc này sẽ làm mất bảo hành của thiết bị. Để đảm bảo thiết bị được cập nhật thành công và tránh gây lỗi thiết bị thì cần tuân thủ các bước từ cài đặt trình điều khiển, sử dụng các công cụ ghi tệp hệ thống từ hệ điều hành, tránh dùng các phần mềm không tin cậy. Nếu không tuân thủ các yêu cần trên thiết bị có thể sẽ bị lỗi và không thể hoạt động sau khi cập nhật.

4.6. Định hướng tiếp theo

Mối quan ngại về an ninh thông tin trên điện thoại di động đang gia tăng. Các bảo vệ ở mức HĐH như Kylin, Saint và Security-by-Contact cung cấp các máy bảo mật cải tiến cho Android và Windows Mobile. Các tiếp cận trên chống lại việc truy cập đến các thông tin nhạy cảm, tuy nhiên khi thông tin được đưa vào ứng dụng thì không thể thêm được bất cứ điều chỉnh nào. Với hệ thống có những màn hình to hơn, các widget đồ họa có thể hỗ trợ người dùng các cách truy cập trực quan hơn. Các HĐH điều khiển luồng thông tin phân quyền (DIFC[1]) cải tiến như Asbestos [11] và HiStar [13] xử lý gán nhãn và truy xuất điều khiển dựa trên mô hình lưới Denning cho bảo mật luồng thông tin. Flume cung cấp các cải tiến tương tự cho các trừu tượng HĐH.

Các công cụ phân tích ứng dụng cho các chính sách thông tin nhạy cảm còn thiếu các chính sách của Oracle và TightLip. Các công cụ nghiên cứu và xử lý các ứng dụng như một hộp đen. Tuy nhiên công cụ phân tích hộp đen này trở nên không hiệu quả khi các ứng dụng mã hóa các thông tin nhạy cảm trước khi gửi đi. Bảo mật luồng thông tin dựa trên sự mở rộng các ngôn ngữ lập trình bởi việc đánh dấu các biến với các thuộc tính bảo mật. Các trình biên dịch dùng các nhãn bảo mật để tạo ra bản ghi bảo mật. Jif [6, 8], Slam [4] và Laminar [10] cung cấp các đảm bảo DIFC do lập trình viên định nghĩa các vùng bảo mật. Tuy nhiên các ngôn ngữ yêu cầu việc phát triển kỹ lưỡng và thường không tương thích với các thiết kế phần mềm kế thừa. Xử lý taint động cung cấp việc giám sát thông tin cho các ứng dụng kế thừa. Hướng tiếp cận dùng để cải thiện tính toàn vẹn của hệ thống, giám sát hoạt động tiếp cận vùng từ xử lý toàn hệ thống dùng các mở rộng phần cứng và các môi trường giả lập để kiểm tra từng xử lý sử dụng biên dịch nhị phân động (DBT[2]) [9, 7]. Hiệu năng và chi phí phần mềm kết hợp với kiểm tra động đưa ra kết quả trong mảng tối ưu. Nếu mã nguồn sẵn có, việc cải thiện hiệu năng có thể thực hiện với chức năng kiểm tra tĩnh.

[1] DIFC: Information Flow Control for Standard OS Abstractions
[2] DBT: Dynamic Binary Translation

Cuối cùng, việc xử lý taint động đã được ứng dụng cho các máy chủ và các trình thông dịch ảo. Công cụ Halder dùng lớp String của Java với kiểm tra taint để ngăn chặn các tấn công SQL injection. WASP cũng làm tương tự, nhưng nó kiểm tra các ký tự riêng để đảm bảo các truy vấn SQL chỉ chứa chuỗi phụ toàn vẹn. Chandra và Franz đưa ra kiểm tra luồng taint với áy ảo Java (JVM [1]) và công cụ mã Java để hỗ trợ xử lý luồng điều khiển.

[1] JVM: Java Virtual Machine

KẾT LUẬN

Luận văn đã cải tiến TaintDroid thành công như mục tiêu đặt ra, tính năng giám sát truy cập các thông tin nhạy cảm trong lịch sự trình duyệt đã hoạt động. Sau khi cải tiến, hệ thống vẫn hoạt động với các luồng xử lý chính như ban đầu. Hiệu năng của hệ thống sau khi cải tiến cũng duy trì ở mức độ cao. Các tài nguyên khi chạy ứng dụng cải tiến ra tăng không đáng kể và không ảnh hưởng nhiều so với hệ thống ban đầu. Phần cải tiến cũng đã đưa ra giải pháp tổng thể cho việc phân tích chính sách truy cập taint. Tuy cải tiến vẫn còn một số hạn chế, nhưng nó cũng mang tính ứng dụng cao và dưới đây sẽ chỉ ra hai mặt trên.

- Tính ứng dụng: Hiện nay, việc người dùng thường xuyên bị thu thập thông tin cá nhân nói chung và các thông tin trong lịch sử trình duyệt là rất phổ biến. Nhất là khi họ cài đặt những ứng dụng không tin cậy từ bên thứ 3. Việc sử dụng trình duyệt để đăng nhập vào các website cũng như mua sắm trực tuyến và cung cấp thông tin đăng nhập hay thẻ tín dụng đã rất phổ biến. Cái tiến thông báo truy xuất trái phép thông tin trên từ lịch sử trình duyệt rất thiết thực. Người dùng có thể biết khi nào các ứng dụng không tin cậy truy xuất các thông tin trên. Nó sẽ góp phần hỗ trợ người dùng tốt hơn trong việc kiểm soát bảo mật thông tin cá nhân trong thời đại bùng nổ Internet hiện nay.
- Các hạn chế: Lịch sử trình duyệt chỉ là một cải tiến cụ thể và có thể áp dụng với các thành phần khác. Các cải tiến cũng đang bị giới hạn do hạn chế TaintDroid chỉ kiểm tra được theo luồng dữ liệu. Nó không thể kiểm tra ở mức luồng điều khiển để có thể tối ưu hóa hiệu năng. Vì vậy, chỉ có thể kiểm tra dữ liệu nhận được khi ứng dụng gửi đi và những hành động khả nghi. Tuy nhiên các ứng dụng có thể sẽ lấy thông tin của người dùng thông qua các luồng điều khiển. Việc kiểm tra luồng điều khiển yêu cầu việc xử lý tĩnh, có nghĩa không thể áp dụng để xử lý các ứng dụng bên thứ 3 mà không có mã nguồn. Hơn nữa, với sự tinh vi như hiện nay, các ứng dụng độc hại thực thụ có khả năng sẽ mã hóa dữ liệu trước khi gửi đi, nên việc kiểm tra luồng dữ liệu và lọc thông tin tại thời điểm này sẽ bị vô hiệu hóa.

TÀI LIỆU THAM KHẢO

1. C. Gibler, J. Crussell, J. Erickson, and H. Chen (2012), *Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In Proceedings of the 5th international conference on Trust and Trustworthy Computing*, pp. 291-307.
2. Google, *Android website: http://www.android.com*
3. Google, *Android source website: https://source.android.com*
4. Heintze,N., Andriecke, J.G (1998), *The SLam Calculus: Programming with Secrecy and Integrity. InProceedings of the Symposium on Principles of Programming Languages (POPL)*, pp. 365-377.
5. J. Kim, Y. Yoon, K. Yi, and J. Shin (2012), *SCANDAL: Static analyzer for detecting privacy leaks in android applications*.
6. Li Li , Alexandre Bartel, Tegawendé F. Bissyand, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, Patrick McDaniel (2015), *IccTA: Detecting Inter-Component Privacy Leaks in Android Apps*.
7. Myers,A.C. (1999), *JFlow: Practical Mostly-Static Information Flow Control. InProceedings of the ACM Symposium on Principles of Programming Langauges (POPL)*.
8. Myers,A.C., Andliskov, B. (2000), *Protecting Privacy Using the Decentralized Label Model.ACM Transactions on Software Engineering and Methodology*, pp. 410-442.
9. Qin,F.,Wang,C., Li,Z., Seopkim,H., Zhou,Y., And Wu, Y. (2006), *LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks. InProceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 135-148.
10. Roy,I.,Porter,D.E.,Bond,M.D.,Mckinley,K.S., Andwitchel,E. (2009), *Laminar: Practical Fine-Grained Decentralized Information Flow Control. InProceedings of Programming Language Design and Implementation*.
11. Vandebogart,S., Efstathopoulos,P., Kohler,E., Krohn,M., Frey,C., Ziegler,D., Kaashoek,F., Morris,R., Andmazi` Eres, D. (2007), *Labels and Event Processes in the Asbestos Operating System. ACM Transactions on Computer Systems (TOCS)*.
12. William Enck, Peter Gilbert, Byung-Gon Chun, Landon P.Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth (2010), *TaintDroid: An*

Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation.

13. Zeldovich,N., Boyd-Wickizer,S., Kohler,E., And Mazi` Eres, D. (2006), *Making Information Flow Explicit in HiStar. In Proceedings of the 7th symposium on Operating Systems Design and Implementation (OSDI)*.

PHỤ LỤC

Trong các nội dung báo cáo ở trên, có một số phần vẫn chưa được miêu tả chi tiết. Nên việc tìm hiểu, xây dựng hay phân tích mã nguồn của cải tiến sẽ gặp không ít khó khăn. Vậy nội dung phần phụ lục dưới đây sẽ miêu tả chi tiết về cải tiến gồm các phần sau.

1. Hướng dẫn xây dựng hệ thống

TaintDroid là kết quả của một nghiên cứu trên nền Android. Hiện nay TaintDroid mới nhất được phát triển trên nền Android 4.3 nên các phần cài đặt hệ thống TaintDroid sẽ chỉ áp dụng cho hệ thống Android phiên bản 4.3. Để phát triển ứng dụng cho Android, có thể sử dụng các PC, laptop sử dụng HĐH Windows, MacOS, Ubuntu và dưới đây là cấu hình cho một laptop chạy HĐH Windows 7 trở lên. Trong khuôn khổ luận văn, việc cải tiến được thực hiện trên một máy ảo VMware chạy HĐH Ubuntu 14.2 và máy chủ chạy HĐH Windows 7. Kết quả cải tiến được cài đặt trên điện thoại di động Google Nenus 4 và dưới đây là chi tiết cấu hình tối thiểu cũng nhưng khuyên dùng cho từng đối tượng:

Bảng 1. Cấu hình máy chính

	Tối thiểu	Khuyên dùng
RAM	6GB	8GB ↗
CPU	2 cores	4 cores ↗
HDD	150GB	250GB ↗
Công nghệ ảo hóa	Yêu cầu	Yêu cầu
HĐH	Windows 7	Windows 7 ↗

Bảng 2. Cấu hình máy ảo

	Tối thiểu	Khuyên dùng
RAM	4GB	6GB ↗
CPU	Dual core	4 cores ↗
HDD	100GB	200GB ↗
Công nghệ ảo hóa	Yêu cầu	Yêu cầu
HĐH	Ubuntu 12.x 64bit	Ubuntu 14.x 64bit ↗

Ngoài việc hỗ trợ cài đặt trên Nexus 4, TaintDroid 4.3 còn có thể cài đặt trên các thiết bị khác như Galaxy Nexus, Nexus 7, Nexus 10. TaintDroid là hệ thống mở rộng của Android, nên nó hoàn toàn phải được chạy trên môi trường tương ứng Android. Nên cần một môi trường phát triển hệ thống Android như khuyên dùng từ webite Google Android[1]. Android thường được xây dựng với HĐH GNU/Linux/Ubuntu hoặc Mac OS và cũng có thể được xây đựng với một máy ảo trên hệ thống không hỗ trợ như Windows. Sau khi lấy được mã nguồn TaintDroid, chúng ta có thể tiến hành xây đựng hệ thống như xây dựng một HĐH được tùy biến.

Về các yêu cầu phần mềm để xây dựng hệ thống, TaintDroid cũng giống như các hệ thống khác chạy trên HĐH Android. Dưới đây là thông số phần mềm cần thiết được rút ra từ quá trình cải tiến, tuy nó không hoàn toàn chính xác tuyết đối nhưng đã được kiểm chứng trong khuôn khổ luận văn:

- Yêu cầu cho Eclipse:
  - Eclipse Mar 2.0 trở lên.
  - Android SDK Tools for Windows 20.xx trở lên.
  - Android Development Tools for Eclipe 20.xx trở lên
  - Eclipe IDE for Java Developers 4.3 trở lên.
  - Android Virtual Devices (AVDs).
- Yêu cầu cho Android Studio:
  - Android Studio 2.0 trở lên.
  - Android SDK Tools for Windows 20.xx trở lên.
  - Android SDK Platform 20.xx trở lên..
  - Intel or ARM Sytem Images.
  - Android Virtual Devices (AVDs).
- GNU/Linux:
  - Android 6.0 (Marshmallow) - AOSP: Ubuntu 14.04 (Trusty)
  - Android 2.3.x (Gingerbread) - Android 5.x (Lollipop): Ubuntu 12.04
  - Android 1.5 (Cupcake) - Android 2.2.x (Froyo): Ubuntu 10.04
- Mac OS (Intel/x86)
  - Android 6.0 (Marshmallow) - AOSP master: Mac OS v10.10 (Yosemite) or later with Xcode 4.5.2 and Command Line Tools
  - Android 5.x (Lollipop): Mac OS v10.8 (Mountain Lion) with Xcode 4.5.2 and Command Line Tools
  - Android 4.1.x-4.3.x (Jelly Bean) - Android 4.4.x (KitKat): Mac OS

[1] Website Google Android http://source.android.com

v10.6 (Snow Leopard) or Mac OS X v10.7 (Lion) and Xcode 4.2 (Apple's Developer Tools)

- Android 1.5 (Cupcake) - Android 4.0.x (Ice Cream Sandwich): Mac OS v10.5 (Leopard) or Mac OS X v10.6 (Snow Leopard) and the Mac OS X v10.5 SDK

- Java Development Kit (JDK): Chú ý không có gói OpenJDK 8 nào hỗ trợ Ubuntu 14.04, gói cho Ubuntu 15.04 cần phải cài đặt thủ công.
- Các gói cài đặt khác:
  - Python 2.6 -- 2.7 từ nguồn https://www.python.org
  - GNU Make 3.81 -- 3.82 từ nguồn http://ftp.gnu.org/gnu/make/; Android 3.2.x (Honeycomb) và các phiên bản cũ hơn cần lấy từ make 3.82 để tránh lỗi.
  - Git 1.7 hoặc phiên bản mới hơn từ website git[1]

2. Các bước thực hiện xây dựng hệ thống

Trước khi thực hiện các bước để xây dựng hệ thống TaintDroid, chúng ta cần đảm bảo HĐH Ubuntu trên máy ảo đã sẵn sàng cho việc thực hiện cài đặt cần thiết thông qua các lệnh sau trên cửa sổ lệnh Terminal theo thứ tự sau đây:

- Cập nhật hệ thống:

```
$ sudo apt-get update
```

- Cài đặt Java phiên bản 6:

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo update-alternatives --config java
```

- Cài đặt Git 2.5 trở nên:

```
$ sudo add-apt-repository ppa:git-core/ppa
$ sudo apt-get install git
$ git config --global user.email chungnn@gmail.com
$ git config --global user.name "Chung Nguyen Nam"
```

- Cài đonfig --

```
$ mkdir ~/bin
$ PATH=~/bin:$PATH
$ curl https://raw.githubusercontent.com/esrlabs/git-repo/master/repo >
~/bin/repo
$ chmod a+x ~/bin/repo
```

- Cài đ a+x ~/bin/repoUbuntu (KVM) cho CPU Intel:

```
$ sudo kvm-ok
$ sudo apt-get install qemu-kvm
```

[1] Website tải GIT mới nhất tại địa chỉ https://git-scm.com/download

- Cài đaptlibstdc++6:

  $ sudo apt-get install gcc-4.8-base=4.8.2-19ubuntu1
  $ sudo apt-get dist-upgrade
  $ sudo apt-get install libstdc++6
  $ sudo apt-get install lib32stdc++6 lib32z1 lib32z1-dev

- Cài đapt-get install

  $ wget http://dl.google.com/android/android-sdk_r24-linux.tgz
  tar -xvzf ~/android-sdk-linux/tools ~home/Downloads/ android-sdk_r24-linux.tgz
  $ cd ~/android-sdk-linux/tools
  $./android

- Chọn các gói sau trên giao diện:
  - Tools\Android SDK Tools
  - Tools\Android SDK Platform-tools
  - Tools\Android SDK Build-tools
  - Android 4.3.1\ ... all packages ..
- Cái đid 4.3.1\ ... all p

  $ gedit ~/.bashrc
  Thêm dòng dưới đây lên trên cùng của tệp vừa mở và lưu lại
  $ export PATH=${PATH}:~/android-sdk-linux/tools
  Logout hệ thống và login trở lại

Sau đây là chi tiết 5 bước chính thực hiện việc xây dựng hệ thống TaintDroid 4.3 trên thiết bị Google Nexus 4. Từ việc lấy mã nguồn, cấu hình hệ thống, biên dịch TaintDroid đến việc cài đặt vào thiết bị.

- Bước 1 - Lấy mã nguồn Android:

  $ mkdir -p ~/tdroid/tdroid-4.3_r1
  $ cd ~/tdroid/tdroid-4.3_r1
  $ repo init -u https://android.googlesource.com/platform/manifest -b android-4.3_r1
  $ repo sync

Kiểm tra kết quả bước 1 bằng cách biên dịch thử hệ thống và không được sửa bất cứ gì trong mã nguồn Android. Đảm bảo biên dịch thành công trước khi chuyển sang bước 2.

$. build/envsetup.sh
$ lunch 1
$ make clean
$ make -j4

- Bước 2 - Lấy mã nguồn TaintDroid: Tải tệp local_manifest.xml đã được sửa đổi từ website chính thức và thay thế cho tệp hiện tại vào tệp .repo/local_manifests/local_manifest.xml sau đó tiến hành lấy mã nguồn về bằng các lệnh sau:

```
$ cd ~/tdroid/tdroid-4.3_r1
$ repo sync --force-sync
$ repo forall dalvik libcore frameworks/base frameworks/native
frameworks/opt/telephony system/vold system/core device/samsung/manta
device/samsung/tuna \packages/apps/TaintDroidNotify -c 'git checkout -b
taintdroid-4.3_r1 --track github/taintdroid-4.3_r1 && git pull'
```

- Bước 3 - Lấy các tệp nhị phân cho thiết bị: Các thiết bị Galaxy Nexus, Nexus 4, Nexus 7 và Nexus 10 yêu cầu các tệp nhị phân riêng không có trong các bản AOSP. Cách lệnh dưới đây sẽ lấy các tệp nhị phân dành cho Nexus 4 có tên mã Mako.

```
$ cd ~/tdroid/tdroid-4.3_r1
$ wget https://dl.google.com/dl/android/aosp/asus-tilapia-jwr66y-
1671e4a8.tgz
$ tar -zxvf asus-tilapia-jwr66y-1671e4a8.tgz
$./extract-asus-tilapia.sh # (view the license and then type "I ACCEPT")
$ wget https://dl.google.com/dl/android/aosp/broadcom-tilapia-jwr66y-
b1271a01.tgz
$ tar -zxvf broadcom-tilapia-jwr66y-b1271a01.tgz
$./extract-broadcom-tilapia.sh
$ tar -zxvf elan-tilapia-jwr66y-61ff5ff6.tgz
$./extract-elan-tilapia.sh
$ tar -zxvf invensense-tilapia-jwr66y-62642635.tgz
$./extract-invensense-tilapia.sh
$ ar -zxvf nvidia-tilapia-jwr66y-b2aa4337.tgz
$ /extract-nvidia-tilapia.sh
$ tar -zxvf nxp-tilapia-jwr66y-1ffb1bc2.tgz
$./extract-nxp-tilapia.sh
$ tar -zxvf widevine-tilapia-jwr66y-777880cf.tgz
$./extract-widevine-tilapia.sh
```

- Bước 4 – Biên dịch TaintDroid:

```
$ cd ~/tdroid/tdroid-4.3_r1
$. build/envsetup.sh
$ lunch full_mako-userdebug
$ make clean
```

```
$ make -j4
$ make bootimage
$ make userdataimage
$ make systemimage
```

- Bước 5 - Ghi vào thiết bị:

```
$ cd ~/tdroid/tdroid-4.3_r1
$ cd out/target/product/mako
$ fastboot flash boot boot.img
$ fastboot flash system system.img
$ fastboot flash userdata userdata.img
```

- Bước 6 (Tùy chọn) - Cài đặt các ứng dụng của Google: Tải tệp "occam-lmy47o-factory-cae68e81.tar" từ các nguồn chính thức trên Internet và tải công cụ miễn phí Nexus Root ToolKit phiên bản 2.0 trở nên. Kết nối thiết bị và sử dụng công cụ này để cài đặt tệp occam-lmy47o-factory-cae68e81.tar.

3. Mã nguồn cải tiến truy cập trình duyệt

Dưới đây là toàn bộ mã nguồn lớp LogBrowserHistory được tạo mới để cải tiến theo dõi truy cập lịch sử trình duyệt.

```
package org.appanalysis;

import android.annotation.SuppressLint;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
   import java.util.Date;
/**
 * ChungNN 2016/08/12
 * A class to handle log of browser history, which application is in black
list.
 */
public class LogBrowserHistory
{
      private static final String LOGTAG =
LogBrowserHistory.class.getSimpleName();
      private static final String FILENAME = String.format("{0}.log",
LogBrowserHistory.LOGTAG.toString());
    private OutputStreamWriter writer = null;
    private FileReader reader = null;
    private ArrayList<HistoryLogEntry> logList = new
ArrayList<HistoryLogEntry>();

      private LogBrowserHistory(){

      }
```

```
    }

    private static LogBrowserHistory instance = new LogBrowserHistory();

    /**
     * Get instance of browser history log
     * @return
     */
  public static LogBrowserHistory getInstance() {
      return instance;
  }

  /**
   * Check if taint data against the access policy
   * @return
   */
  public boolean IsValidAccess(String data){
        return (data.indexOf("username") != -1
                    || data.indexOf("password") != -1
                    || data.indexOf("credit") != -1);
  }

  /**
   * Check if an application is in black list
   * @param appName
   * @return
   */
    public boolean IsInBlackList(String appName){
        return IsInBlackList(appName, "");
    }

    /**
     * Check if an application is in black list by name and IpAddress
     * @param appName
     * @param ipAddress
     * @return
     */
    @SuppressLint("DefaultLocale")
    public boolean IsInBlackList(String appName, String ipAddress)
    {
        if(logList == null || logList.isEmpty())
            return false;

        HistoryLogEntry log;
        for(int i = 0; i < logList.size(); i++)
        {
            log = logList.get(i);
            if(appName != "" &&
log.AppName().toLowerCase().equals(appName))
                return true;
            if(ipAddress != "" &&
log.IpAddress().toLowerCase().equals(ipAddress))
                return true;
            if(appName != "" && ipAddress != ""
                    &&
(log.AppName().toLowerCase().equals(appName) ||
log.IpAddress().toLowerCase().equals(ipAddress)))
            {
                return true;
            }
        }
        return false;
```

```
	}

	/**
	 * Opens browser history log
	 * @throws IOException
	 */
	public void open() throws IOException {
		LoadLog();
	}

	/**
	 * Closes browser history log
	 * @throws IOException
	 */
	public void close() throws IOException {
		SaveLog();
	}

	/**
	 * Add an application into black list.
	 * @param appName
	 * @param note
	 * @throws IOException
	 */
	public void AddBlackList(String appName, String note) throws
IOException{
		AddBlackList(appName, "", note);
	}

	/**
	 * Add an application into black list of browser history access.
	 * @param appName
	 * @param ipAddress
	 * @param note
	 * @return
	 * @throws IOException
	 */
	public void AddBlackList(String appName, String ipAddress, String
note) throws IOException{
		String logTime = new Date().toString();
		HistoryLogEntry log = new HistoryLogEntry(logTime, appName,
ipAddress, note);
		logList.add(log);
	}

	/**
	 * Loads data from log file into memory
	 */
	private void LoadLog()
	{
		try
		{
			if(logList == null)
			logList = new ArrayList<HistoryLogEntry>();
			BufferedReader br = null;
			try
			{
				br = new BufferedReader(reader);
				String line = br.readLine();
				HistoryLogEntry log;
				while(line != null)
				{
```

```
                                    //Line format
"logTime@#appName@#ipAddress@#note"
                                    String[] arr = line.split("\\@#", -1);
                                    log = new HistoryLogEntry(arr[0], arr[1],
arr[2], arr[3]);
                                    logList.add(log);
                                    line = br.readLine();
                              }
                        }
                        catch(IllegalArgumentException e)
                        {

                        }
                        finally
                        {
                              if(br != null)
                                    br.close();
                        }

                  }
                  catch( IOException e )
                  {

                  }
            }

            /**
             * Saves the log data from memory into log file
             * @throws IOException
             */
            private void SaveLog() throws IOException
            {
                  if(logList != null && !logList.isEmpty())
                  {
                        if(writer == null)
                              writer = new FileWriter(FILENAME, false); //No
append
                        HistoryLogEntry log;
                        BufferedWriter output = null;
                        try
                    {
                        output = new BufferedWriter(writer);
                        for(int i = 0; i < logList.size(); i++)
                              {
                                    log = logList.get(i);
                                    //Line format
"logTime@#appName@#ipAddress@#note"

      output.write(String.format("{0}@#{1}@#{2}@#{3}", log.LogTime(),
log.AppName(), log.IpAddress(), log.Note()));
                              }
                    }
                        catch ( IOException e )
                        {
                    }
                        finally {
                      if ( output != null )
                        output.close();
                      if(writer != null)
                                    writer.close();
                    }
                  }
```

```
        }

        /**
         * Class present to the browser history log entry
         * @author ChungNN
         *
         */
        private class HistoryLogEntry
    {
        private String logTime;
        private String appName;
        private String ipAddress;
        private String note;

        public HistoryLogEntry(String logTime, String appName, String
ipAddress, String note)
        {
              this.logTime = logTime;
              this.appName = appName;
              this.ipAddress = ipAddress;
              this.note = note;
        }
        //...
        public String LogTime()
        {
            return this.logTime;
        }

        public String AppName()
        {
              return this.appName;
        }

        public String IpAddress()
        {
              return this.ipAddress;
        }

        public String Note()
        {
              return this.note;
        }
     }
    }
```

4. Mã nguồn cải tiến tổng quát

Như đã miêu tả sơ bộ về mã nguồn ở chương 3 về giải pháp tổng quát, toàn bộ mã nguồn của cải tiến đều được tạo mới thành các tệp PrivacyRule, PrivacyRecord và PrivacyAnalyzer tương ứng với các lớp, sau đây là toàn bộ mã nguồn của module này:

- Định nghĩa luật

```
package org.appanalysis;

/**
 * The privacy record describes detail of violation.
```

```
 * @author ChungNN. 2017/02/15
 */
public class PrivacyRecord {
    private boolean isViolated = false;
    private ViolatedLevel violatedLevel = ViolatedLevel.Low;
    private String description;

    public boolean isViolated() {
          return isViolated;
    }

    public void setViolated(boolean isViolated) {
          this.isViolated = isViolated;
    }

    public String getDescription() {
          return description;
    }

    public void setDescription(String description) {
          this.description = description;
    }

    public ViolatedLevel getViolatedLevel() {
          return violatedLevel;
    }

    public void setViolatedLevel(ViolatedLevel violatedLevel) {
          this.violatedLevel = violatedLevel;
    }

    public enum ViolatedLevel {
          Low, Medium, High;
    }
}
```

- Định nghĩa bản ghi

```
package org.appanalysis;

/* A class to define the privacy rule
* @author ChungNN. 2017/02/21
*/
public class PrivacyRule {
    private EnumRuleName ruleName;
    private String[] ruleKeys;

    public EnumRuleName getRuleName() {
          return ruleName;
    }
    public void setRuleName(EnumRuleName ruleName) {
          this.ruleName = ruleName;
    }
    public String[] getRuleKeys() {
          return ruleKeys;
    }
    public void setRuleKeys(String[] ruleKeys) {
          this.ruleKeys = ruleKeys;
    }

    public PrivacyRule(EnumRuleName name, String[] keys)
    {
          this.ruleName = name;
```

```
            this.ruleKeys = keys;
    }

    public enum EnumRuleName {
            Location, AddressBoox, Microphone, Phone_Number, GPS_Location,
            NETBased_Location, Last_Known_Location, Camera, Accelerometer,
SMS,
            IMEI, IMSI, ICCID, Device_Sserial_Number,
User_Account_Information,
            Browser_History
    }
}
```

- Phân tích luật

```
/**
 * Analyze the taint data is matched to rules
 * @author ChungNN, 2017/03/15
 */
public class PrivacyAnalyzer {
    private static PrivacyAnalyzer instance = new PrivacyAnalyzer();
    private static Hashtable<Integer, String> ttable = new
Hashtable<Integer, String>();
    static {
        // ttable.put(new Integer(0x00000000), "No taint");
        ttable.put(new Integer(0x00000001), "Location");
        ttable.put(new Integer(0x00000002), "Address Book
(ContactsProvider)");
        ttable.put(new Integer(0x00000004), "Microphone Input");
        ttable.put(new Integer(0x00000008), "Phone Number");
        ttable.put(new Integer(0x00000010), "GPS Location");
        ttable.put(new Integer(0x00000020), "NET-based Location");
        ttable.put(new Integer(0x00000040), "Last known Location");
        ttable.put(new Integer(0x00000080), "camera");
        ttable.put(new Integer(0x00000100), "accelerometer");
        ttable.put(new Integer(0x00000200), "SMS");
        ttable.put(new Integer(0x00000400), "IMEI");
        ttable.put(new Integer(0x00000800), "IMSI");
        ttable.put(new Integer(0x00001000), "ICCID (SIM card
identifier)");
        ttable.put(new Integer(0x00002000), "Device serial number");
        ttable.put(new Integer(0x00004000), "User account information");
        ttable.put(new Integer(0x00008000), "Browser history");
    }

    /**
     * Get the instance of class
     * @return
     */
    public static PrivacyAnalyzer getInstance() {
        return instance;
    }

    /**
     * Analyze privacy from the queue message is matched with rules
     * @param msg
     * @param rules
     * @return
     */
    public PrivacyRecord Analyze(String msg, ArrayList<PrivacyRule>
rules)
    {
```

```
        if(msg.equals(null) || msg.equals("") || rules.size() == 0)
              return null;
        return AnalyzeTaint(msg, rules);
    }

    /**
     * Analyze privacy from the queue message is matched with rules
     * @param msg
     * @param rules
     * @return
     */
    private PrivacyRecord AnalyzeTaint(String msg, ArrayList<PrivacyRule>
rules)
    {
          String taint = get_taint(msg);
        if(taint.equals(""))
          return null;

        PrivacyRecord record = new PrivacyRecord();
          boolean isValid = false;
          for (PrivacyRule rule : rules) {
                if(rule != null)
                {
                      isValid = IsValideRule(rule, msg);
                      if(!isValid)
                            break;
                }
          }

          record.setViolated(!isValid);
          if(record.isViolated())
          {
                String ip = get_ipaddress(msg);
                record.setDescription(String.format("This taint is
volidated to access {0}", "..."));
          }
          return record;
    }

    /**
     * Process a rule is matched with queue message
     * @param rule
     * @param msg
     * @return
     */
    private boolean IsValideRule(PrivacyRule rule, String msg)
    {
          if(rule == null || msg.equals(""))
                return false;

          EnumRuleName ruleName = rule.getRuleName();
          String data = get_data(msg);
          String ip = get_ipaddress(msg);

          switch (ruleName)
          {
                case Location:
                      break;
                      //TODO implement logic for specific rule...
                case AddressBoox:
                      break;
                case Microphone:
```

```
                break;
            case Phone_Number:
                break;
            case GPS_Location:
                break;
            case NETBased_Location:
                break;
            case Last_Known_Location:
                break;
            case Camera:
                break;
            case Accelerometer:
                break;
            case SMS:
                break;
            case IMEI:
                break;
            case IMSI:
                break;
            case ICCID:
                break;
            case Device_Sserial_Number:
                break;
            case User_Account_Information:
                break;
            case Browser_History:
                break;
            default:
                break;
        }
        return true;
    }

    /**
     * Get the ip address from queue message
     * @param msg
     * @return
     */
    private String get_ipaddress(String msg) {
        Pattern p = Pattern.compile("\\((.+)\\) ");
        Matcher m = p.matcher(msg);

        if(m.find() && m.groupCount() > 0) {
          String result = m.group(1);
          // remove trailing junk
          if (result.contains(")"))
                result = result.substring(0,result.indexOf(")")-1);
            return result;
        }
        else {
            return null;
        }
    }

    /**
     * Get taint from taint data
     * @param msg
     * @return
     */
    private String get_taint(String msg) {
        // match hex digits
        Pattern p = Pattern.compile("with tag 0x(\\p{XDigit}+) ");
```

```
        Matcher m = p.matcher(msg);

        if(m.find() && m.groupCount() > 0) {

            String match = m.group(1);

            // get back int
            int taint;
            try {
                taint = Integer.parseInt(match, 16);
            }
            catch(NumberFormatException e) {
                return "";
            }

            if(taint == 0x0) {
                return "";
            }

            // for each taint
            ArrayList<String> list = new ArrayList<String>();
            int t;
            String tag;

            // check each bit
            for (int i=0; i<32; i++) {
                t = (taint>>i) & 0x1;
                tag = ttable.get(new Integer(t<<i));
                if(tag != null) {
                    list.add(tag);
                }
            }

            // build output
            StringBuilder sb = new StringBuilder("");
            if(list.size() > 1) {
                for(int i = 0; i < list.size() - 1; i++) {
                    sb.append(list.get(i) + ", ");
                }
                sb.append(list.get(list.size() - 1));
            }
            else {
                if(!list.isEmpty()) {
                    sb.append(list.get(0));
                }
            }

            return sb.toString();
        }
        else {
            return "";
        }
    }

    /**
     * Get the taint data from queue message
     * @param msg
     * @return
     */
    private String get_data(String msg) {
        int start = msg.indexOf("data=[") + 6;
        return msg.substring(start);
    }
}
```