BAN CƠ YẾU CHÍNH PHỦ

HỌC VIỆN KỸ THUẬT MẬT MÃ

BÁO CÁO

Nghiên cứu các giải pháp phát hiện mã độc trên thiết bị Android

Môn học : Cơ sở an toàn thông tin

Sinh viên thực hiện:

Nguyễn Thị Kim Huế

AT13CLC0110

Giảng viên hướng dẫn:

Trần Anh Tú

Hà Nội - 2019

MỤC LỤC

MỞ ĐẦU

Điện thoại thông minh đã trở nên rất phổ biến trong vài năm trở lại đây. Trong sự phát triển của thị trường di động thông minh, Android một nền tảng mã nguồn mở của Google đã trở thành một trong những hệ điều hành di động phổ biến nhất. Android chủ yếu được sử dụng trên điện thoại thông minh và máy tính bảng. Điện thoại thông minh ngày càng được chấp nhận và sử dụng bởi rất nhiều yếu tố như các thiết bị điện thoại thông minh có khả năng cung cấp các dịch vụ như ngân hàng, mạng xã hội, các ứng dụng văn phòng và hầu hết tất cả mọi thứ đều có thể làm với điện thoại thông minh. Điện thoại thông minh được trang bị một số tính năng như kết nối wifi, gọi điện, lưu trữ dữ liệu, định vị toàn cầu(GPS),...

Đi đôi với sự phát triển của hệ điều hành Android thì số lượng mã độc phát triển trên hệ điều hành này cũng ngày một tăng cao. Năm 2012 số lượng mã độc phát hiện mới trên nền tảng Android là 214.327 mẫu đến năm 2016 đã tăng lên 3.246.284 mẫu mã độc được phát hiện mới [6]. Điều này dẫn đến các phần mềm phòng chống mã độc trên Android cũng cần cải tiến về phương pháp, Kỹ thuật. Đã có rất nhiều nghiên cứu tập trung vào việc phát hiện phần mềm độc hại trên Android. Một trong những phương pháp phổ biến bao gồm các phương pháp dựa trên chữ ký, trích chữ ký từ phần mềm độc hại mẫu. Mặc dù nó có hiệu quả để phát hiện phần mềm độc hại đã biết, nhưng nó không đủ để phát hiện phần mềm độc hại chưa biết. Ngoài ra còn một số phương pháp dựa trên việc phân tích hoạt động mạng của các phần mềm. Phương pháp này thực hiện giám sát lưu lượng truy cập mạng của một ứng dụng mẫu và cố gắng phát hiện phần mềm độc hại bằng cách so sánh với danh sách blacklist của DNS và địa chỉ IP. Phương pháp này không thể phát hiện phần mềm độc hại chưa được xác định, vì blacklist được tạo ra từ các hoạt động của phần mềm độc hại đã được biết đến. Isohara [3] trình bày một phương pháp để phát hiện phần mềm độc hại bằng cách phân tích các thuộc tính của các tập tin trong các mẫu ứng dụng. Mặc dù cách tiếp cận này có thể phát hiện một số phần mềm độc hại không xác định mà không bị phát hiện bởi blacklist hoặc phương pháp phân tích dựa trên chữ ký, chi phí phân tích phụ thuộc vào số lượng tệp trong mẫu phân tích. Enck và cộng sự [9] đề xuất một phương pháp để ngăn chặn việc cài đặt các ứng dụng có quyền nguy hiểm hoặc intent filter (một cơ chế để thực hiện hợp tác giữa các ứng dụng Android). Tuy nhiên, phương pháp có thể dẫn đến phát hiện không chính xác, bởi vì thông tin được sử dụng trong phương pháp không đủ để phân biệt phần mềm độc hại từ các ứng dụng lành tính. Ngoài ra còn phương pháp phân tích mã độc dựa trên việc phân tích các lời gọi API trong tệp tin smali như trong nghiên cứu của Wu và cộng sự [14]. Tuy nhiên, việc thực hiện phương pháp trên sẽ nảy sinh vấn đề đó là chi phí phân tích rất lớn nó tùy thuộc vào số lượng tệp tin và kích thước của tệp tin trong ứng dụng ban đầu.

Bài tiểu luận được thực hiện nhằm mục đích tìm hiểu về các phương pháp phát hiện mã độc dựa trên phân tích tĩnh.

CHƯƠNG 1: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH ANDROID

1.1. Tổng quan về sự phát triển của hệ điều hành android

Điện thoại thông minh đã trở nên rất phổ biến trong vài năm trở lại đây. Số lượng người dùng smartphone trên toàn thế giới được dự báo sẽ tăng từ 3,4 tỷ vào năm 2015 lên khoảng 6,4 tỷ trong năm 2021 [10]. Trong sự phát triển của thi trường di động thông minh, Android một nền tảng mã nguồn mở của Google đã trở thành một trong những hệ điều hành di động phổ biến nhất.

Hình 1.1 là thị phần thị trường di động của Android, iOS và các hệ điều hành di động khác [20]:

Hình 0.1: Thị phần của các hệ điều hành di động quý 4 năm 2012 (Theo thống kê của Gartner vào quý 4 năm 2012) [20]

Operating System	4Q16 Units	4Q16 Market Share (%)	4Q15 Units	4Q15 Market Share (%)
Android	352,669.9	81.7	325,394.4	80.7
iOS	77,038.9	17.9	71,525.9	17.7
Windows	1,092.2	0.3	4,395.0	1.1
BlackBerry	207.9	0.0	906.9	0.2
Other OS	530.4	0.1	887.3	0.2
Total	**431,539.3**	**100.0**	**403,109.4**	**100.0**

Hình 0.2: Thị phần của các hệ điều hành di động quý 4 năm 2016 (Theo thống kê của Gartner vào quý 4 năm 2016)

Sự gia tăng đột ngột của các ứng dụng trên điện thoại thông minh gây ra mối lo ngại về bảo mật cho người dùng. Điện thoại di động đã trở thành mục tiêu của các nhà phát triển các ứng dụng độc hại. Android trở nên phổ biến bởi vì nó là hệ điều hành mã nguồn mở và có một số tính năng cơ bản như các middleware(là phần

mềm có nhiệm vụ kết nối các thành phần của mềm hoặc ứng dụng lại với nhau) giữa các máy ảo và một số ứng dụng cơ bản như máy tính, lịch, trình duyệt,...

1.2. Cấu trúc tệp tin trong hệ điều hành Android

Hệ thống quản lý tập tin trong android là được phát triển từ linux nên có nhiều đặc điểm giống với hệ thống quản lý tập tin trên linux.

1.2.1. Tổng quan về hệ thống tệp tin trên Android

Trong Android, các tệp tin được tổ chức thành các thư mục, theo mô hình phân cấp. Tham chiếu đến một tệp tin bằng tên và đường dẫn. Các câu lệnh thao tác tệp tin cho phép thực hiện các chức năng như dịch chuyển, sao chép toàn bộ thư mục cùng với các thư mục con chứa trong nó.

Có thể sử dụng các ký tự, dấu gạch dưới, chữ số, dấu chấm và dấu phẩy dể đặt tên tệp tin. Không duợc bắt đầu một tên tệp tin bằng dấu chấm hay chữ số. Những ký tự khác như '/', '?', '*', là ký tự dặc biệt được dành riêng cho hệ thống. Chiều dài của tên tệp tin có thể tới 256 ký tự. Trong hệ diều hành Android có sự phân biệt tên tệp tin chữ hoa và chữ thuờng, điều dó có nghĩa là trong cùng 1 thư mục có thể tồn tại những tệp tin có tên là File, FILE, file.. và chúng là những tệp tin khác nhau. Tất cả các tệp tin trong Android có chung cấu trúc vật lý là chuỗi các byte (byte stream). Cấu trúc thống nhất này cho phép Android áp dụng khái niệm tệp tin cho mọi thành phần dữ liệu trong hệ thống. Thư mục cũng như các thiết bị được xem như tệp tin. Chính việc xem mọi thứ như các tệp tin cho phép Android quản lý và chuyển đổi dữ liệu một cách dễ dàng. Một thư mục chứa các thông tin về thư mục, được tổ chức theo một định dạng đặc biệt. Các thành phần được xem như các tệp tin, chúng được phân biệt dựa trên kiểu tệp tin: tệp tin bình thường, tệp tin thư mục, tệp tin kiểu ký tự, và tệp tin kiểu khối.

1.2.2. Các kiểu tệp tin trên android

Trong nhiều hệ điều hành như window, người ta phân biệt rõ tập tin(file) và thư mục (folder) hay directory là 2 thành phần khác hẳn nhau. Tuy nhiên trên hệ điều hành Android (cũng như Linux) thì coi directory cũng là tệp tin và nó là một loại tệp tin đặc biệt. Thực tế còn một số loại tệp tin nữa có thể liệt kê theo bảng sau [1]:

Bảng 0.1: Bảng liệt kê một số kiểu tệp tin trong Linux

Chữ cái biểu diễn	Kiểu tệp tin
D	Thư mục (directory)
b	Tệp tin kiểu khối (block-type special file)
C	Tệp tin kiểu ký tự (character-type special file)
L	Liên kết tượng trưng (symbolic link)
P	Tệp tin đường ống (pipe)
S	Socket

-	Tệp tin bình thường (regular file)

1.2.3. Tổ chức quyền sở hữu và quyền hạn trên tệp tin

Tương tự trên hệ thống linux, trên hệ điều hành android, một tệp tin có thể liên kết với một người sử dụng và một nhóm người sử dụng. Sự liên kết đó là một tập hợp các quyền hạn truy cập bao gồm quyền được phép đọc (read), được phép ghi (write) và được phép thực thi (execute).

Cụ thể như sau: Một tệp tin sẽ có những quyền hạn tương ứng với 9 ký tự theo mẫu sau : với 3 ký tự "r,w,x" nghĩa là có quyền tương ứng với ký tự viết tắt đó, "-" nghĩa là không có quyền hạn đó(bảng 1.2):

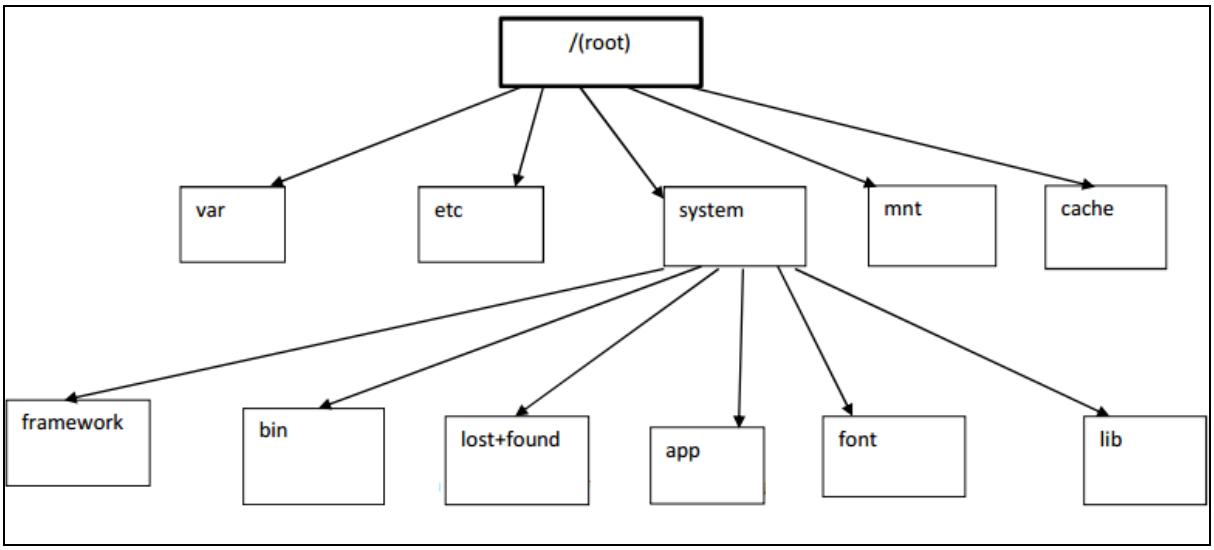
Bảng 0.2: Một số quyền làm việc với tệp tin

Chủ nhân tệp tin (Owner)			Nhóm tài khoản sở hữu tệp tin (Owner Group)			Những người không thuộc nhóm sở hữu tệp tin(Other)		
r/-	w/-	x/-	r/-	w/-	x/-	r/-	w/-	x/-

- 3 ký tự đầu tiên là quyền hạn chủ nhân tệp tin
- 3 ký tự giữa là quyền hạn của nhóm tài khoản sở hữu tệp tin
- 3 ký tự cuối là quyền hạn của những người không thuộc nhóm sở hữu tệp tin.

1.2.4. Cây thư mục trên hệ điều hành Android

Thư mục (hay có thể gọi là file) root là thư mục gốc của tất cả các tệp tin thư mục còn lại. Dưới nó có chứa một số tệp tin thư mục hệ thống. Mỗi thư mục (trừ thư mục root) đều có một thư mục cha chứa nó, bản thân nó cũng có thể có nhiều tệp tin thư mục con. Cấu trúc đó có thể mô tả bằng một cây thư mục có dạng như sau [1]:



Hình 0.3: Cấu trúc một cây thư mục đơn giản

Giới thiệu một vài thư mục tiêu biểu [18] :

- /(root) : là thư mục gốc, là thư mục duy nhất không có thu mục cha.
- / mnt : thư mục chứa thiết bị lưu động (removeable).
- /system : chứa những thành phần cơ bản nhất của hệ thống.
- /ect : chứa những tệp tin cấu hình của hệ thống, nó cực kỳ quan trọng vì sự hoạt động của hệ thống đều bị chi phối ở những tệp tin cấu hình này.
- /system/lost+found : chứa những tập tin bị mất lúc khởi động máy.
- /system/font : chứa những font chữ hiển thị được.
- /system/lib : chứa các thư viện để các phần mềm hoạt động (các phần mềm viết bằng ngôn ngữ java).
- /system/app : chứa các tệp tin apk của phần mềm. (Các tệp tin cài đặt ứng dụng, tương tự như MSI trong window hay dev trong Linux).
- /system/bin : chứa các chương trình nội trú của hệ thống.

1.3. Vấn đề về mã độc trên hệ điều hành Android

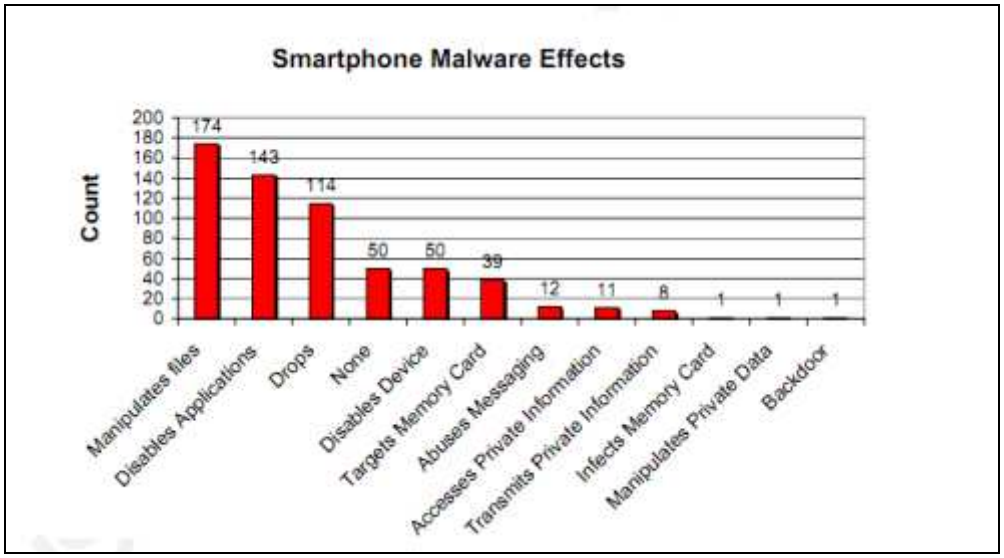
Kiến trúc của Android được tạo thành từ các phần sau:

- Một hạt nhân Linux(Linux Kernel) có hỗ trợ multiprocess(nhiều tiến trình) và multuthreads(nhiều luồng). Mỗi ứng dụng có một Linux ID riêng và chạy một tiến trình riêng biệt. Hai ứng dụng có cùng một ID có thể trao đổi dữ liệu với nhau.
- Một số thư viện mã nguồn mở
- Môi trường Android run-time, trong một máy ảo Dalvik chạy một ứng dụng với định dạng nhị phân dex.
- Application Framework bao gồm các Java interface. Lớp này bao gồm Android NDK và SDK.
- Các ứng dụng được cài đặt

Các mô hình an toàn cho thiết bị android dựa trên cơ chế phân quyền và sanbox. Mỗi ứng dụng sẽ chạy trong máy ảo Dalvik riêng của mình với một ID duy nhất được gán cho ứng dụng đó. Điều này ngăn cản các ứng dụng sẽ cản trở thông tin và dữ liệu của ứng dụng khác.

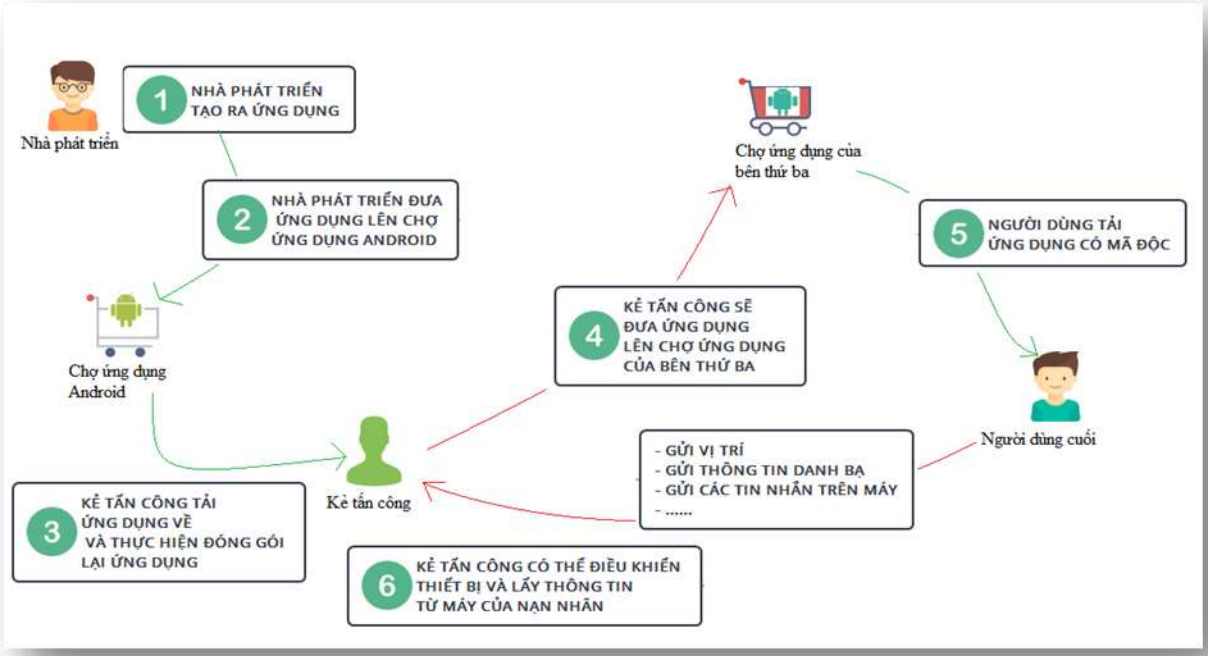
Các nhà phát triển phần mềm của bên thứ ba sẽ tạo ra các ứng dụng mới và đưa chúng lên chợ ứng dụng của Android. Điều này sẽ cho phép người dùng có thể truy cập vào hàng ngàn ứng dụng, do đó cần để người sử dụng hoàn toàn tin tưởng vào ứng dụng trước khi cài đặt chúng. Vì lí do đó mà mọi ứng dụng cần đưa ra các quyền đòi hỏi trong quá trình cài đặt. Người dùng có quyền chấp nhận hoặc từ chối tất cả các quyền đó, trong trường hợp này, quá trình cài đặt ứng dụng sẽ bị hủy.

Tuy nhiên, có nhiều ứng dụng độc hại vẫn có mặt trên chợ ứng dụng của Android. Do đó nó sẽ trở thành điều cần thiết cho Google kiểm tra các ứng dụng thường xuyên và làm sạch thị trường ứng dụng di động cho Android bằng cách xóa bỏ phần mềm độc hại. Ngoài phần mềm độc hại ra còn một số loại tấn công khác trên các ứng dụng di động như: tấn công lừa đảo, kết nối http không an toàn, sử dụng dữ liệu local,...
Dựa trên các mẫu mã độc đã được công bố, hình 1.3 dưới đây sẽ cho thấy số lượng mã độc và tác động của chúng [5]:



Hình 0.4: Tác động của malware trên điện thoại thông minh

Từ đồ thị trên ta có thể thấy hầu hết các mã độc trên di động đã thành công trong việc kiểm soát các tập tin và vô hiệu hóa ứng dụng khác. Trong trường hợp của hệ điều hành Android, các ứng dụng được đăng trên chợ ứng dụng của Android, các ứng dụng được xuất bản trên chợ ứng dụng Android là nơi người dùng có thể xem và tải các ứng dụng đó về và cài đặt trên máy. Kẻ tấn công thường xáo trộn lại ứng dụng và xuất bản lại chúng. Người dùng tải về các ứng dụng đó mà không biết rằng ứng dụng đó không phải là bản gốc và phần mềm độc hại đó đã được cài đặt trên thiết bị của họ.
Một trong những cách mà một kẻ tấn công có thể lôi kéo người dùng tải về các phần mềm độc lại là đóng gói lại các ứng dụng bằng cách sử dụng các công cụ dịch ngược. Kẻ tấn công sẽ thay đổi mã lệnh của chương trình kết hợp với các mã độc hại và đóng gói lại ứng dụng sau đó phát hành ra thị trường ứng dụng di động. Người dùng thường không thể phân biệt giữa phần mềm độc hại và phần mềm hợp pháp ban đầu. Quá trình trên sẽ được mô tả ở sơ đồ sau [15]:

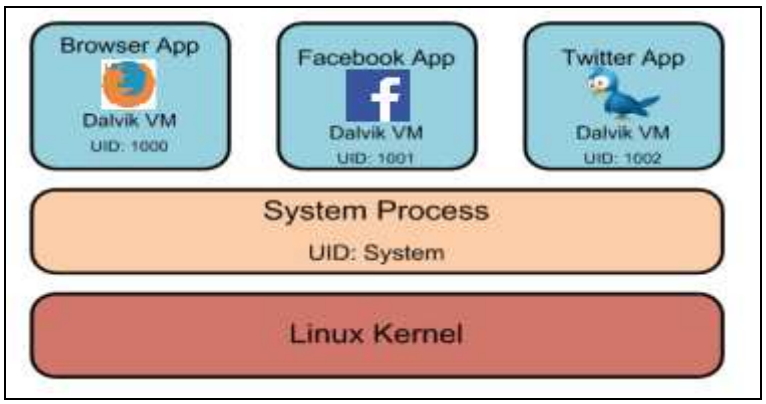


Hình 0.5: Quá trình đóng gói lại một ứng dụng

Số lượng của mã độc và phần mềm gián điệp ngày càng tăng nhanh. Do đó việc phân tích một ứng dụng có an toàn hay không trước khi thực hiện cài đặt là rất cần thiết. Điều đó sẽ giúp bảo vệ cho sự an toàn cho dữ liệu của người dùng và tổ chức.

1.4. Mô hình bảo mật của android và các nguy cơ mất an toàn

Mô hình bảo mật trên Android được thiết kế để không cho ứng dụng được phép thực hiện bất kì hoạt động xấu nào làm ảnh hưởng đến các ứng dụng khác, người sử dụng hoặc hệ điều hành. Mỗi ứng dụng chạy trong một tiến trình riêng của nó và do đó Android được coi là hệ thống đa xử lí. Trong linux các nhóm và người dùng sử dụng ID gán cho các ứng dụng và thực hiện áp dụng an ninh cho hệ thống và cho các ứng dụng ở mức tiến trình. Một mức an ninh nữa là sử dụng cơ chế cấp quyền để hạn chế về hoạt động của một tiến trình cụ thể [12].
Trong mô hình bảo mật của Android chủ yếu dựa trên cơ chế bảo vệ và cho phép. Mỗi ứng dụng sẽ chạy trong một máy ảo Dalvik cụ thể với một ID duy nhất được gán, có nghĩa là mã nguồn của ứng dụng này chạy độc lập với mã nguồn của ứng dụng khác. Như vậy ứng dụng không được cấp quyền thì sẽ không thể truy cập vào tập tin của ứng dụng khác [12].



Hình 0.6: Mô hình bảo mật của Android

Mỗi ứng dụng Android đã được ký kết với một chứng chỉ có khóa riêng để biết chủ sở hữu của các ứng dụng là duy nhất. Điều này cho phép các tác giả của ứng dụng sẽ được xác định nếu cần thiết. Khi một ứng dụng được cài đặt trong điện thoại nó được gán một ID người dùng, do đó tránh được nó tự ảnh hưởng đến các ứng dụng khác bằng cách tạo một sandbox cho nó. ID người dùng này là vĩnh viễn trên đó thiết bị và các ứng dụng có cùng ID người dùng được phép chạy trong một tiến trình duy nhất. Đây là một cách để đảm bảo rằng một ứng dụng độc hại có thể không truy cập/thỏa hiệp dữ liệu của ứng dụng chính hãng [17].
Nó là bắt buộc đối với một ứng dụng để liệt kê tất cả các tài nguyên nó sẽ truy cập trong khi cài đặt. Các điều khoản được yêu cầu của một ứng dụng, trong quá trình cài đặt,cần được sự chấp thuận của người dùng tương tác hoặc dựa trên kiểm tra cùng với chữ ký của các ứng dụng [17].
Quyền của ứng dụng Android là cần thiết ở các giai đoạn khác nhau trong vòng đời của một ứng dụng, cụ thể là [17]:

- Vào thời điểm một lời gọi hệ thống để ngăn chặn các ứng dụng từ việc thực hiện các chức năng cụ thể mà không mong muốn.
- Khởi đầu của một hoạt động để tránh các hoạt động của các ứng dụng khác khởi xướng.
- Quyền quản lí người có thể gửi, nhận tin nhắn.
- Khởi động dịch vụ.
- Thực hiện hoạt động hoặc truy cập một nội dung trên máy.

Có thể chia mức độ an ninh trên thiết bị Android thành 4 cấp độ với tên cụ thể như sau:

- Mức độ bình thường(Normal): trong trường hợp này không có quyền nào yêu cầu sự cho phép của người dùng, do đó các quyền bình thường sẽ được trao cho ứng dụng.
- Mức độ nguy hiểm(Dangerous): những quyền này theo yêu cầu của một ứng dụng tới người sử dụng trong quá trình cài đặt. Người dùng có thể chấp nhận tất cả các quyền hoặc từ chối tất cả. Sự từ chối của các quyền sẽ chấm dứt quá trình cài đặt.

- Signature: những điều khoản được thừa nhận bởi hệ thống cung cấp và các ứng dụng yêu cầu có chứng chỉ tương tự.
- Signature System: nó cũng tương tự như Signature nhưng chỉ áp dụng cho các ứng dụng hệ thống.

Vô số phương pháp bảo mật tích hợp cho Android làm cho nó an toàn chỉ khi nó được xử lý bởi người sử dụng có trách nhiệm và hiểu những ảnh hưởng của một số quyền được yêu cầu bởi các ứng dụng khác nhau.
Bên cạnh đó hệ điều hành Android cũng bộc lộ ra một số lỗi của mình khiến cho người dùng có thể bị tấn công và bị đánh cắp thông tin cá nhân.

Một số lỗ hổng an ninh trên Android:

- Lộ thông tin qua các bản ghi log: Việc truy cập log trên Android được cung cấp thông qua Log API, nó có thể hiển thị thông tin qua câu lệnh "logcat". Ứng dụng với quyền READ_LOG có thể đọc thông tin log, trong log có thể chứ các thông tin nhạy cảm của người dùng [21].
- Sử dụng thẻ nhớ ngoài: một ứng dụng có quyền ĐỌC(READ) hoặc GHI(WRITE) thẻ nhớ ngoài thì nó có khả năng đọc toàn bộ thông tin được lưu trên đó từ bất kì ứng dụng nào [21].
- Wi-Fi Sniffing: điều này có thể làm xáo trộn các dữ liệu đang được truyền từ một thiết bị như nhiều trang web và các ứng dụng không có các biện pháp an ninh nghiêm ngặt. Ứng dụng không mã hóa dữ liệu và do đó nó có thể được chặn bởi một kẻ biết lắng nghe trên đường truyền không an toàn [21].
- Khai thác mạng: khai thác các lỗ hổng phần mềm hiện tại của hệ điều hành hoặc phần mềm đối với các mạng di động hoặc mạng địa phương(local). Những khai thác thường không yêu cầu bất kỳ sự can thiệp từ người sử dụng vì thế nó được coi là nguy hiểm nhất [21].

Các vấn đề an ninh lớn đối với điện thoại thông minh đang chạy trên hệ điều hành Android là người sử dụng không có kiến thức hoặc bất cẩn cài đặt phần mềm độc hại hoặc bị các cuộc tấn công khác như tấn công lừa đảo. Các vấn đề an ninh quan trọng thứ hai là một số ứng dụng hợp pháp lại có lỗ hổng có thể bị khai thác.

CHƯƠNG 2: NGHIÊN CỨU KỸ THUẬT PHÁT HIỆN MÃ ĐỘC DỰA TRÊN PHÂN TÍCH TĨNH

2.1. Một số phương pháp lây nhiễm mã độc vào thiết bị Android

Có một số phương pháp mà các thiết bị Android có thể bị nhiễm phần mềm độc hại. Sau đây là bốn phương pháp khác nhau mà phần mềm độc hại có thể được cài đặt trên điện thoại:

- **Đóng gói lại ứng dụng hợp pháp:** Đây là một trong những phương pháp phổ biến nhất được sử dụng bởi những kẻ tấn công. Họ có thể tìm và tải về ứng dụng phổ biến trên thị trường, sử dụng các công cụ dịch ngược, thêm các đoạn mã độc hại và sau đó đóng gói lại thành các ứng dụng mới và đưa ra thị trường ứng dụng Android chính thức hoặc của bên thứ ba. Người dùng có thể dễ dàng cài đặt các ứng dụng này do bị dụ dỗ để tải về và cài đặt các ứng dụng bị nhiễm mã độc. Đã có khoảng 86,0% ứng dụng hợp pháp bị đóng gói lại bao gồm cả các mã độc hại sau khi phân tích hơn 1.200 mẫu phần mềm độc hại Android [22].
- **Khai thác các lỗ hổng trên ứng dụng Android**: Có thể là một lỗi trong các ứng dụng của người dùng. Những kẻ tấn công có thể sử dụng lỗ hổng này để thỏa hiệp điện thoại và cài đặt phần mềm độc hại trên thiết bị.
- **Ứng dụng giả mạo:** Nó cũng đã được phát hiện ra rằng có những ứng dụng giả mạo là các phần mềm độc hại cho phép kẻ tấn công truy cập vào thiết bị di động của bạn. Những kẻ tấn công tải lên trên thị trường ứng dụng các ứng dụng giả mạo là các phần mềm chứa mã độc của kẻ tấn công. Ví dụ: Kẻ tấn công tải lên một phần mềm có tên Facebook nhưng thực tế đó lại là phần mềm do kẻ tấn công viết và có chứa mã độc trong đó.
- **Cài đặt từ xa**: Các phần mềm độc hại có thể được cài đặt từ xa lên điện thoại của người dùng. Nếu kẻ tấn công có thể lấy được các thông tin của người sử dụng và vượt qua chúng trên chợ ứng dụng, sau đó trong trường hợp này, các phần mềm độc hại sẽ được cài đặt vào thiết bị mà không cần sự can thiệp từ phía người dùng. Ứng dụng này sẽ chứa mã độc hại cho phép kẻ tấn công truy cập dữ liệu cá nhân như danh sách liên lạc, tin nhắn,....

2.2. Các kỹ thuật phân tích mã độc trên Android

a) Kỹ thuật phân tích động

Phân tích động đôi khi cũng được gọi là phân tích hành vi, được sử dụng để phân tích và nghiên cứu hành vi của phần mềm độc hại. Sau đó nghiên cứu các cách phần mềm độc hại tương tác với hệ thống, dịch vụ, thu thập dữ liệu, thực hiện

kết nối mạng, mở cổng dịch vụ,...

Trong giai đoạn này, các tập tin apk được cài đặt trên một thiết bị mô phỏng(hoặc thiết bị thật) để quan sát hành vi của ứng dụng. Một báo cáo được tạo ra và các hành vi được so sánh với các kết quả trong phần phân tích tĩnh.

Phân tích động được thực hiện bằng cách chạy các ứng dụng trên các thiết bị thực tế và giả lập. Với các đầu vào khác nhau, chẳng hạn như lưu lượng mạng và truy cập tập tin được theo dõi. Nexus S được sử dụng để thử nghiệm trực tiếp, vì nó là một thiết bị với hệ điều hành Android mặc định từ Google mà không cần bất kỳ phần mềm nào của bên thứ ba. Điều này sẽ cho một cái nhìn tổng quát hơn để thử nghiệm. Tcpdump được sử dụng để theo dõi lưu lượng mạng, và DDMS được sử dụng để phân tích tiến trình và số liệu khác.

b) Kỹ thuật phân tích tĩnh

Phân tích tĩnh được gọi là phân tích mã nguồn, được sử dụng để phân tích mã nguồn của phần mềm độc hại. Mục đích chính là để biết chính xác các đoạn mã độc hại được nhúng trong mã nguồn của ứng dụng.

Giai đoạn này bao gồm việc phân tích các tập tin apk và tất cả các nội dung của tập tin đó. Các nội dung được truy cập được bằng cách chuyển đổi chúng thành một hình thức có thể đọc được. Các classes được chuyển đổi từ DEX thành các classes Java, và các tập tin nhị phân XML, AndroidManifest.xml, được chuyển đổi sang XML có thể đọc được. Sau khi chuyển đổi thành công, đang xem xét bất kỳ hành vi đáng ngờ nào trong mã nguồn. Điều này được thực hiện bằng cách thông qua các quyền yêu cầu của các ứng dụng trong mã nguồn,... sau đó, một nhật ký được lập ra để ghi lại tất cả các hành vi nhận thấy rằng có thể được coi là độc hại.

c) So sánh các kỹ thuật phân tích mã độc trên Android

Bảng 0.1: So sánh giữa hai Kỹ phân tích

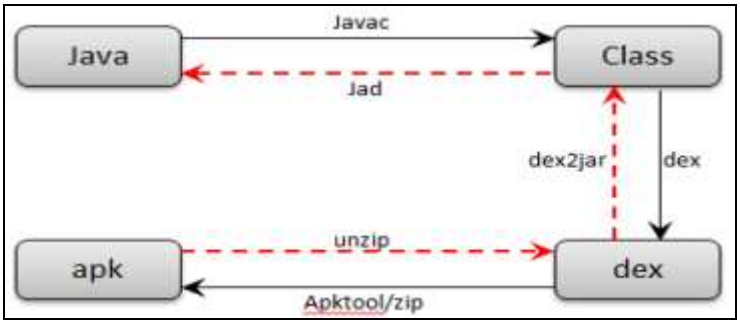
	Phân tích tĩnh	Phân tích động
Chế độ phân tích	Phân tích ứng dụng khi ứng dụng đó không thực thi.	Phân tích ứng dụng khi ứng dụng đang ở chế độ thực thi.

Phân tích mã độc	- Sử dụng các công cụ dịch ngược để lấy mã nguồn của ứng dụng từ tệp tin APK(dex2jar, APKtool,...). - Kiểm tra quyền của ứng dụng thông qua tệp tin AndroidManifest.xml. - Phân tích dựa trên các đặc điểm nghi ngờ bởi các họ nhất định. - Các lời gọi hệ thống thông qua API được phân tích nhằm phát hiện hành vi nguy hiểm của ứng dụng.	- Phân tích dựa trên các tính năng, hành vi của các ứng dụng khi chạy các ứng dụng. - Phân tích dựa trên việc kiểm tra các cuộc gọi hệ thống và đường dẫn thực thi. - Thông tin bị rò rỉ trong quá trình phân tích và tiêu hao năng lượng được giám sát để phát hiện hành vi nguy hiểm của các ứng dụng.
Hạn chế	- Trong trường hợp mã nguồn bị làm rối sẽ giảm hiệu quả của phân tích. - Mã nguồn không thể được phân tích trong trường hợp phân tích bytecode. - Không thể phát hiện ra mẫu mã độc mới.	- Cần nhiều thời gian và công sức để phân tích ứng dụng trong các khoảng thời gian chạy khác nhau.

2.3. Nghiên cứu kỹ thuật dịch ngược

Dịch ngược là quá trình phân tích mã code của phần mềm để kiểm tra và phát hiện các lỗ hổng hoặc lỗi của phần mềm. Dịch ngược là quá trình tạo ra mã nguồn từ mã thực thi. Kỹ thuật này sử dụng để kiểm tra các chức năng của chương trình hoặc được thực hiện để bỏ qua các bước kiểm tra các cơ chế bảo mật, v.v. Do đó kỹ thuật dịch ngược còn được sử dụng trong quá trình chỉnh sửa mã nguồn chương trình để chương trình thực hiện theo ý đồ của người dịch ngược mong muốn.

Quá trình dịch ngược được sử dụng để phân tích mã độc cho các ứng dụng Android. Đây là một quá trình dịch ngược một ứng dụng để hiểu cách làm việc và chức năng của ứng dụng bằng cách phân tích mã nguồn và gỡ lỗi ứng dụng đó.



Hình 0.3: Quá trình xây dựng và dịch ngược tệp tin APK

Một khi ứng dụng khi được tải về trên điện thoại từ Google Market, tệp tin .apk có sẵn. Vì vậy, trước hết, các tập tin nên được un-packaging bằng cách sử dụng lệnh như “unzip”. Sau đó, các tập tin và thư mục sau đây sẽ được tìm thấy trong thư mục giải nén.

Trong phân tích tĩnh tôi sẽ sử dụng hai cách sau để phân tích một ứng dụng android:

- Sử dụng ApkTool để disassemble ứng dụng Android và thực hiện phân tích smali code bằng cách sử dụng Notepad++.
- Sử dụng Dex2Jar để chuyển mã nguồn từ tệp tin .dex sang dạng code java và sau đó sử dụng Notepad++ hoặc JDGui để phân tích code java.

Áp dụng các kĩ thuật phân tích tĩnh vào phân tích mã độc trên Android

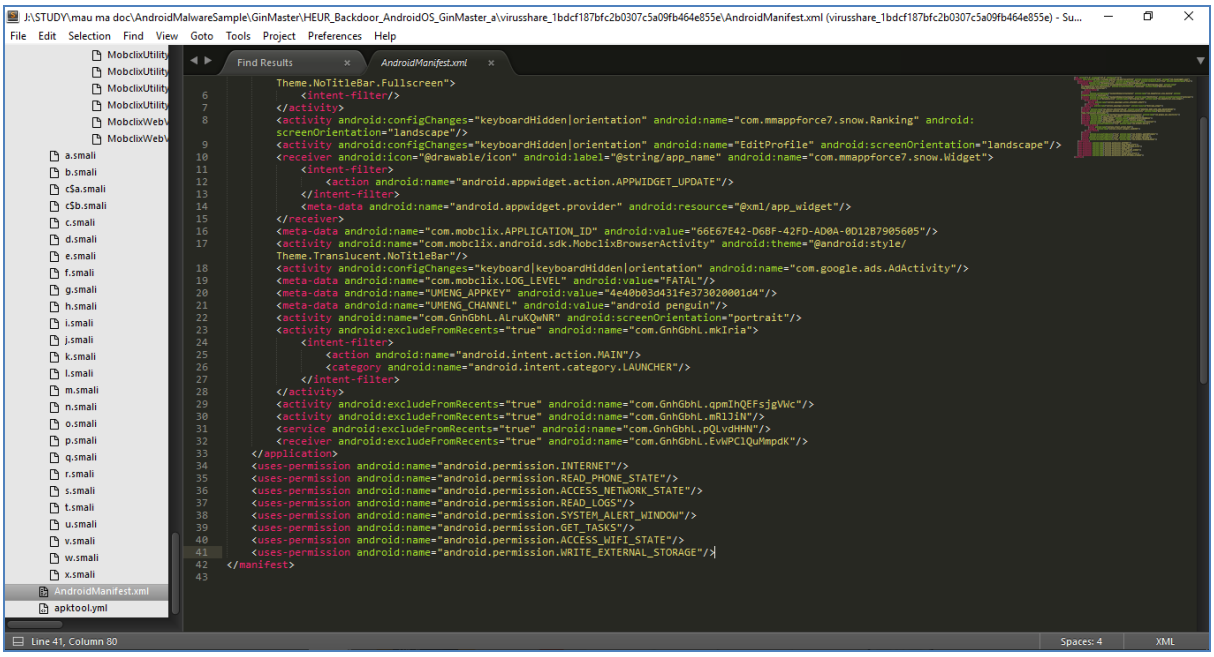
Mẫu sử dụng trong phân tích:

https://drive.google.com/open?id=0B1Eckcvi1T6hbE94MG9KTHdkN1E

Sử dụng phương pháp dịch ngược 1.

Sử dụng ApkTool để phân tích tập tin .apk.

Sử dụng Notepad++ hoặc một công cụ chỉnh sử văn bản để xem tệp tin manifest như hình 3.1.



Hình 0.1: Sử dụng SublineText để xem tệp tin manifest

Mở tệp tin AndroidManifest.xml ta thu được các quyền ứng dụng yêu cầu:

android.permission.SYSTEM_ALERT_WINDOW

android.permission.ACCESS_NETWORK_STATE

android.permission.READ_PHONE_STATE

android.permission.READ_CONTACTS

android.permission.WRITE_EXTERNAL_STORAGE

android.permission.INTERNET

android.permission.GET_TASKS

android.permission.READ_LOGS

Phân tích tập tin smali ta thu được các thông tin như sau:

smali\com\GnhGbhL\f.smali

```
387     sput-object v0, Lcom/GnhGbhL/f;->L:Ljava/lang/String;
388
389     const-string v0, "java -jar signapk.jar testkey.x509.pem testkey.pk8 xxx1.apk"
390
391     sput-object v0, Lcom/GnhGbhL/f;->M:Ljava/lang/String;
```

Ứng dụng thực hiện lấy thông tin đường dẫn về bộ nhớ trong và các tệp tin được người dùng tải xuống.

```
440 .method private static a()Ljava/lang/String;
441     .locals 3
442
443     const-string v0, "mounted"
444
445     invoke-static {}, Landroid/os/Environment;->getExternalStorageState()Ljava/lang/String;
446
447     move-result-object v1
448
449     invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
450
451     move-result v0
452
453     if-eqz v0, :cond_1
454
455     sget-object v0, Landroid/os/Environment;->DIRECTORY_DOWNLOADS:Ljava/lang/String;
456
457     invoke-static {v0}, Landroid/os/Environment;->getExternalStoragePublicDirectory(Ljava/lang/String;)Ljava/io/File;
458
459     move-result-object v0
460
461     invoke-virtual {v0}, Ljava/io/File;->exists()Z
462
```

```
457     invoke-static {v0}, Landroid/os/Environment;->getExternalStoragePublicDirectory(Ljava/lang/String;)Ljava/io/File;
458
459     move-result-object v0
460
461     invoke-virtual {v0}, Ljava/io/File;->exists()Z
462
463     move-result v0
464
465     if-eqz v0, :cond_0
466
467     sget-object v0, Landroid/os/Environment;->DIRECTORY_DOWNLOADS:Ljava/lang/String;
468
469     invoke-static {v0}, Landroid/os/Environment;->getExternalStoragePublicDirectory(Ljava/lang/String;)Ljava/io/File;
470
471     move-result-object v0
472
473     invoke-virtual {v0}, Ljava/io/File;->toString()Ljava/lang/String;
474
475     move-result-object v0
476
477     :goto_0
478     return-object v0
479
480     :cond_0
481     :try_start_0
482     invoke-static {}, Landroid/os/Environment;->getExternalStorageDirectory()Ljava/io/File;
483
484     move-result-object v0
485
486     invoke-virtual {v0}, Ljava/io/File;->getPath()Ljava/lang/String;
487
488     move-result-object v0
```

Cố gắng lấy thông tin về bộ nhớ.

```
524     invoke-virtual {v1}, Ljava/io/File;->mkdirs()Z
525
526     invoke-virtual {v1}, Ljava/io/File;->getPath()Ljava/lang/String;
527     :try_end_0
528     .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
529
530     move-result-object v0
531
532     goto :goto_0
533
```

Thực hiện tạo thư mục mới

```
1594    invoke-virtual {v2}, Ljava/io/File;->createNewFile()Z
1595
1596    new-instance v1, Ljava/io/BufferedWriter;
1597
1598    new-instance v3, Ljava/io/FileWriter;
1599
1600    invoke-direct {v3, v2}, Ljava/io/FileWriter;-><init>(Ljava/io/File;)V
1601
1602    invoke-direct {v1, v3}, Ljava/io/BufferedWriter;-><init>(Ljava/io/Writer;)V
1603
1604    invoke-virtual {v1, p0}, Ljava/io/BufferedWriter;->write(Ljava/lang/String;)V
1605
1606    invoke-virtual {v1}, Ljava/io/BufferedWriter;->close()V
1607    :try_end_2
1608    .catch Ljava/lang/Exception; {:try_start_2 .. :try_end_2} :catch_0
1609
1610    goto :goto_0
```

Tạo tệp tin mới

```
1771    const-string v3, "phone"
1772
1773    move-object/from16 v0, p0
1774
1775    invoke-virtual {v0, v3}, Landroid/content/Context;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;
1776
1777    move-result-object v3
1778
1779    check-cast v3, Landroid/telephony/TelephonyManager;
1780
1781    invoke-virtual {v3}, Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;
1782
1783    move-result-object v8
1784
1785    invoke-virtual {v3}, Landroid/telephony/TelephonyManager;->getSubscriberId()Ljava/lang/String;
1786
1787    move-result-object v7
1788
1789    invoke-virtual {v3}, Landroid/telephony/TelephonyManager;->getSimSerialNumber()Ljava/lang/String;
1790
1791    move-result-object v6
1792
1793    invoke-virtual {v3}, Landroid/telephony/TelephonyManager;->getLine1Number()Ljava/lang/String;
1794
1795    move-result-object v5
1796
1797    invoke-virtual {v3}, Landroid/telephony/TelephonyManager;->getNetworkType()I
1798
1799    move-result v3
1800
1801    invoke-static {v3}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;
1802
1803    move-result-object v4
```

Lấy thông tin về điện thoại

Như ta có thể thấy trong đoạn code trên ứng dụng đang cố gắng lấy các thông tin của điện thoại như: DeviceId, SubscriberId, SimSerialNumber, Line1Number,

NetworkType.

```
1807    invoke-interface {v12, v3, v8}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/String;)
        Landroid/content/SharedPreferences$Editor;
1808
1809    sget-object v3, Lcom/GnhGbhL/f;->d:Ljava/lang/String;
1810
1811    invoke-interface {v12, v3, v7}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/String;)
        Landroid/content/SharedPreferences$Editor;
1812
1813    sget-object v3, Lcom/GnhGbhL/f;->f:Ljava/lang/String;
1814
1815    invoke-interface {v12, v3, v9}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/String;)
        Landroid/content/SharedPreferences$Editor;
1816
1817    sget-object v3, Lcom/GnhGbhL/f;->i:Ljava/lang/String;
1818
1819    invoke-interface {v12, v3, v6}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/String;)
        Landroid/content/SharedPreferences$Editor;
1820
1821    sget-object v3, Lcom/GnhGbhL/f;->j:Ljava/lang/String;
1822
1823    invoke-interface {v12, v3, v5}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/String;Ljava/lang/String;)
        Landroid/content/SharedPreferences$Editor;
1824    :try_end_1
1825    .catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_0
1826    .catchall {:try_start_1 .. :try_end_1} :catchall_0
```

Ghi thông tin vào bộ nhớ chia sẻ dạng key:value

Trong tệp tin MobclickAgen.smali

```
912     :try_start_1
913     const-string v1, "type"
914
915     const-string v2, "update"
916
917     invoke-virtual {v0, v1, v2}, Lorg/json/JSONObject;->put(Ljava/lang/String;Ljava/lang/Object;)Lorg/json/JSONObject;
918
919     const-string v1, "appkey"
920
921     invoke-virtual {v0, v1, p2}, Lorg/json/JSONObject;->put(Ljava/lang/String;Ljava/lang/Object;)Lorg/json/JSONObject;
922
923     invoke-virtual {p1}, Landroid/content/Context;->getPackageManager()Landroid/content/pm/PackageManager;
924
925     move-result-object v1
926
927     invoke-virtual {p1}, Landroid/content/Context;->getPackageName()Ljava/lang/String;
928
929     move-result-object v2
930
931     const/4 v3, 0x0
932
933     invoke-virtual {v1, v2, v3}, Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/content
        PackageInfo;
934
935     move-result-object v1
936
937     iget v1, v1, Landroid/content/pm/PackageInfo;->versionCode:I
938
939     invoke-virtual {p1}, Landroid/content/Context;->getPackageName()Ljava/lang/String;
940
941     move-result-object v2
942
943     const-string v3, "version_code"
944
945     invoke-virtual {v0, v3, v1}, Lorg/json/JSONObject;->put(Ljava/lang/String;I)Lorg/json/JSONObject;
946
947     const-string v1, "package"
```

Thu thập thông tin ứng dụng:

- phiên bản code(version_code)
- package
- appkey
- update

```
1881    const-string v4, "logcat"
1882
1883    invoke-virtual {v3, v4}, Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
1884
1885    const-string v4, "-d"
1886
1887    invoke-virtual {v3, v4}, Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
1888
1889    const-string v4, "-v"
1890
1891    invoke-virtual {v3, v4}, Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
1892
1893    const-string v4, "raw"
1894
1895    invoke-virtual {v3, v4}, Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
1896
1897    const-string v4, "-s"
1898
1899    invoke-virtual {v3, v4}, Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
1900
1901    const-string v4, "AndroidRuntime:E"
1902
1903    invoke-virtual {v3, v4}, Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
1904
1905    const-string v4, "-p"
1906
1907    invoke-virtual {v3, v4}, Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
1908
1909    invoke-virtual {v3, v1}, Ljava/util/ArrayList;->add(Ljava/lang/Object;)Z
1910
1911    invoke-static {}, Ljava/lang/Runtime;->getRuntime()Ljava/lang/Runtime;
1912
1913    move-result-object v4
```

Truy cập logcat

```
1980    move-result-object v1
1981
1982    const-string v2, "logcat -c"
1983
1984    invoke-virtual {v1, v2}, Ljava/lang/Runtime;->exec(Ljava/lang/String;)Ljava/lang/Process;
1985    :try_end_1
1986    .catch Ljava/lang/Exception; {:try_start_1 .. :try_end_1} :catch_0
1987
```

Ứng dụng cố gắng xóa log.

```
2069    const-string v3, "Failed to clear log"
2070
2071    invoke-static {v2, v3}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I
2072
2073    invoke-virtual {v1}, Ljava/lang/Exception;->printStackTrace()V
2074    :try_end_2
2075    .catch Ljava/lang/Exception; {:try_start_2 .. :try_end_2} :catch_1
2076
2077    goto :goto_1
```

Kết luận:

Bằng kết quả phân tích ta có thể thấy ứng dụng là một ứng dụng gián điệp. Khi thực hiện cài đặt vào máy ứng dụng sẽ thực hiện một số công việc như sau:

- Thực hiện lấy các thông tin về điện thoại của nạn nhân: DeviceId, SubscriberId, SimSerialNumber, Line1Number, NetworkType.
- Thực hiện tạo mới tệp tin để lưu trữ các thông tin thu thập được.
- Lấy thông tin các ứng dụng trên máy như: Phiên bản code, package, appkey

và tình trạng update của ứng dụng.

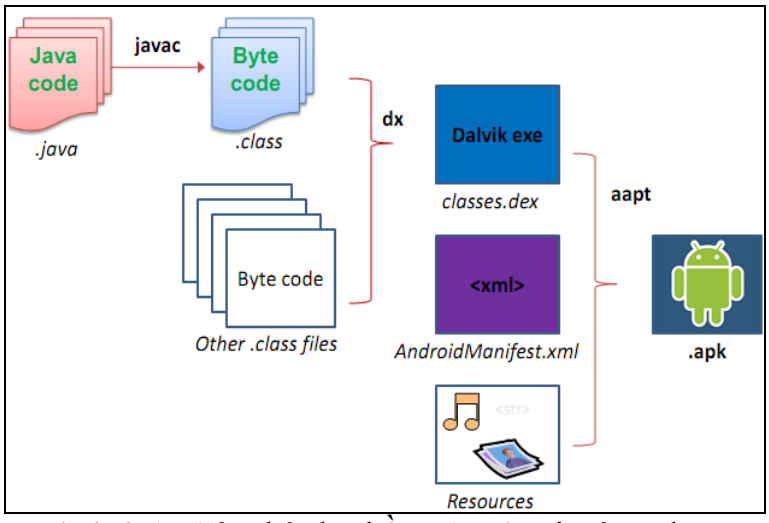
- Sau khi thực hiện lấy thông tin ứng dụng cố gắng truy cập log và thực hiện xóa log nhằm xóa dấu vết.
- Với những thông tin thu được ứng dụng có thể sẽ gửi chúng về máy chủ của hacker nhằm thu thập thông tin của người dùng nhằm cho mục đích tấn công của chúng.

CHƯƠNG 3: ÁP DỤNG CÁC KỸ THUẬT PHÂN TÍCH TĨNH VÀO PHÂN TÍCH MÃ ĐỘC TRÊN ANDROID

3.1. Phát hiện mã độc trên Android bằng cách phân tích tệp tin Manifest

3.1.1. Cơ sở lí thuyết

Một ứng dụng Android bao gồm các phần như sau:



Hình 3.1: Các thành phần của tệp tin ứng dụng

AndroidManifest.xml: tệp tin cung cấp thông tin cần thiết để ứng dụng hoạt động bình thường với hệ thống Android, trong đó hệ thống sẽ đọc tệp tin này trước khi có thể chạy các đoạn mã khác của ứng dụng. Trong đó có một số thông tin như: các quyền mà ứng dụng yêu cầu, các API tối thiểu để ứng dụng hoat động, danh sách thư viện ứng dụng cần,...

Classes.dex: mã nguồn Java được biên dịch để chạy trong máy ảo Dalvik.

Resources: bao gồm hai phần chính, một thư mục res chứa các tài nguyên cần thiết cho ứng dụng không được biên dịch trước như: ảnh, String,... và tệp tin resources.arsc chứa tài nguyên đã được biên dịch trước.

META-INF: thư mục này chứa một số metadata như chứng chỉ của ứng dụng, tệp tin manifest của ứng dụng java.

Lib: các thư viện được biên dịch sẵn phù hợp với từng nền tảng phần cứng.

Assets: chứa các tài nguyên mà ứng dụng có thể truy cập thông qua AssetManager.

Kết quả phân tích tệp tin từ 30 ứng dụng Android độc hại và 30 ứng dụng Android lành tính [16].

Bảng 0.1: Bảng kích thước trung bình các tệp tin (đơn vị: KB)

	Tệp tin smali	Tệp tin mã nguồn	Tệp tin manifest
20 mẫu mã độc	4503	3670	5
20 mẫu lành tính	2614	1407	4

Bảng 0.2: Bảng số lượng tệp tin trung bình

	Tệp tin smali	Tệp tin mã nguồn	Tệp tin manifest
20 mẫu mã độc	583	342	1
20 mẫu lành tính	256	102	1

Trong tiểu luận này tôi nghiên cứu phương pháp phát hiện mã độc trên Android dựa vào các đặc trưng của tệp tin manifest. Mỗi ứng dụng Android phải có tệp tin manifest, trong đó trình bày thông tin thiết yếu về ứng dụng. Phương pháp đề xuất của chúng tôi dựa trên phân tích đặc trưng của tệp tin Android manifest và có hiệu quả để phát các phần mềm độc hại đã được xác định và phần mềm độc hại chưa được xác định. Hơn nữa, chi phí cho phương pháp này rất thấp, bởi vì phương pháp này chỉ phân tích tệp tin manifest. Tệp tin manifest có kích thước rất nhỏ so với kích thước của các tệp tin như Resource hay smali. Phương pháp này sử dụng các kỹ thuật dịch ngược để trích xuất thông tin từ mã nguồn của ứng dụng.

3.1.2. Các bước triển khai phương pháp

Phần mềm độc hại được phát hiện theo các bước sau:

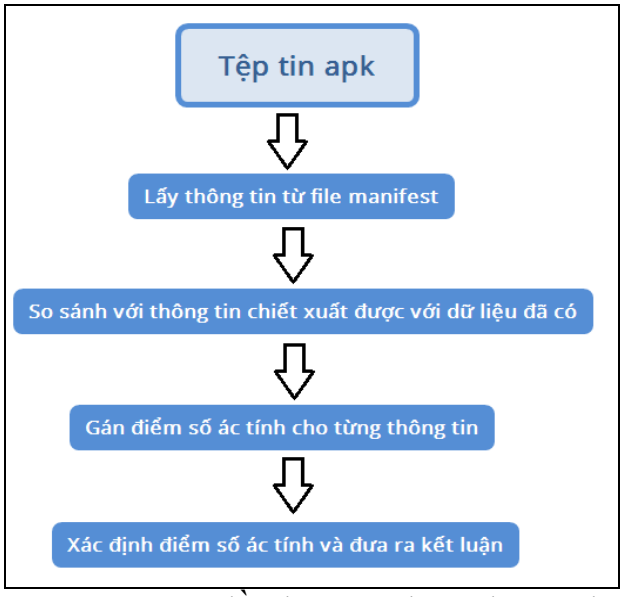
Bước 1: Trích xuất thông tin cụ thể trong tệp tin manifest.

Bước 2: So sánh thông tin trích xuất được với dữ liệu đã có.

Bước 3: Sau đó tính điểm số ác tính cho các thông tin trích xuất được.

Bước 4: Dựa vào điểm số ác tính để phân loại ứng dụng: Là lành tính hay chứa mã độc

Phương pháp phân tích sẽ như sau:



Hình 3.2: Sơ đồ phương pháp phân tích

Tệp tin manifest có các thông tin về các ứng dụng Android, chẳng hạn như phiên bản của ứng dụng, tên của package, các permission(quyền) phải có, và cấp độ(level) API. Định dạng của tệp tin manifest giống hệt nhau trong cả ứng dụng lành tính và phần mềm độc hại. Tuy nhiên, ở đây có sự khác biệt nhất định trong các đặc tính của một số mục thông tin. Bằng thực nghiệm cũng như đọc các tài liệu liên quan. Với kết quả thực hiện điều tra trên 30 mẫu lành tính và 30 mẫu phần mềm độc hại, tổng cộng là 60 mẫu [16]. Tôi đã lựa chọn cụ thể ra một mục thông tin hiển thị trên nhiều loại phần mềm độc hại khác biệt so với các ứng dụng lành tính. Dưới đây là một số thông tin được trích xuất từ tệp tin manifest để sử dụng trong phương pháp này:

(1) Permission

(2) Intent filter (action)

(3) Intent filter (category)

(4) Process name

Trong phương pháp này, một số danh sách các keyword được biên soạn cho một ứng dụng. Một ứng dụng là lành tính hay là ứng dụng độc hại thì chuỗi trong tệp tin manifest đều được ghi lại trong một danh sách với các keyword tương ứng. Ở đây sẽ có 4 kiểu danh sách keyword [16]:

(1) Permission, (2) Intent filter (action), (3) Intent filter (category), và (4) Process name.

Một ứng dụng Android cơ bản mặc định sẽ không có quyền nào liên kết với nó, có ghĩa là nó không thể thực hiện bất cứ điều gì ảnh hưởng xấu tới trải nghiệm người dùng hoặc dữ liệu trên thiết bị. Để sử dụng các tính năng được bảo vệ của thiết bị, người lập trình sẽ phải sử dụng thẻ <use-permission> trong tệp tin manifest để liệt kê các quyến mà ứng dụng muốn sử dụng.

Ví dụ một ứng dụng muốn yêu cầu quyền nhận tin nhắn SMS thì sẽ thực hiện như sau:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.android.app.myapp" >
   <uses-permission android:name="android.permission.RECEIVE_SMS" />
   ...
</manifest>
```

Vì vậy để liệt kê các quyền ứng dụng yêu cầu trong tệp tin manifest ta sẽ liệt kê các quyền được yêu cầu trong thẻ <use-permission>.

Một Intent filter (action) sẽ được định nghĩa trong tệp tin manifest của ứng dụng với cú pháp:

```
<action android:name="string" />
```

Đồng thời các action sẽ chỉ được chứa trong các <intent- filter> vì vậy để lấy ra các thuộc tính Intent filter (action) áp dụng cho phương pháp trên ta sẽ liệt kê toàn bộ các action có trong thẻ <action> của tệp tin manifest.

Một Intent filter (category) sẽ được định nghĩa trong tệp tin manifest của ứng dụng với cú pháp:

```
<category android:name="string" />
```

Tương tự như action, category chỉ được chứa trong các <intent- filter> vì vậy để lấy ra các thuộc tính Intent filter (category) áp dụng cho phương pháp trên ta sẽ liệt kê toàn bộ các action có trong thẻ < category> của tệp tin manifest.

Một process trong tệp tin manifest sẽ được định nghĩa bằng cú pháp:

```
<android:process ="string" />
```

Để xác định các process trong tệp tin manifest ta sẽ dựa vào các thẻ android:process để liệt kê ra các process được sử dụng trong ứng dụng.

Dưới đây là danh sách các Permission trong tệp tin Manifest của các mẫu mà chúng tôi phân tích. Mỗi bảng được chia làm bảng được chia làm ba cột: Số thứ tự, Tên Permission, Số lần xuất hiện.

Với Tên Permission: là tên của quyền đó xuất hiện trong tệp tin manifest.

Số lần xuất hiện: Đếm tổng số lần xuất hiện của quyền đó trong tất cả các mẫu được phân tích.

Đối với 40 mẫu lành tính:

Bảng 0.3: Danh sách các quyền yêu cầu trong ứng dụng lành tính

Số thứ tự	Tên Permission	Số lần xuất hiện
1	INTERNET	20
2	WRITE_EXTERNAL_STORAGE	13
3	ACCESS_NETWORK_STATE	11
4	WAKE_LOCK	9
5	READ_PHONE_STATE	7
6	VIBRATE	9
7	ACCESS_WIFI_STATE	5
8	USE_CREDENTIALS	5

9	MANAGE_ACCOUNTS	4
10	SET_WALLPAPER	4

Đối với 40 mẫu mã độc:

Bảng 0.4: Danh sách các quyền yêu cầu trong mã độc

Số thứ tự	Tên Permission	Số lần xuất hiện
1	READ_PHONE_STATE	35
2	INTERNET	32
3	SEND_SMS	28
4	WRITE_EXTERNAL_STORAGE	26
5	ACCESS_NETWORK_STATE	25
6	RECEIVE_SMS	25
7	READ_SMS	23
8	ACCESS_WIFI_STATE	19
9	WRITE_SMS	17
10	READ_CONTACTS	17

Kết quả trên cho thấy xuất hiện một số quyền phổ biến. Các con số này cho thấy rằng các quyền liên quan đến gửi tin nhắn (SMS), chẳng hạn như SEND_SMS, RECEIVE_SMS và READ_SMS thường được sử dụng bởi các mẫu mã độc. Các quyền này sẽ được cho vào danh sách các quyền nguy hiểm. Ta sẽ tiến hành tương tự để tìm ra các keyword phân loại cho (2) Intent filter (action), (3) Intent filter (category), và (4) Process name. Các danh sách keyword thu được sẽ được sử dụng trong quá trình đánh giá độ ác tính của các mẫu phân tích.

Kết quả thu thập từ các mẫu:

(Danh sách 1) Permission(quyền) nguy hiểm được liệt kê trong bảng 2.6.

Bảng 0.5: Các permission nguy hiểm

Số thứ tự	Permission
1	READ_SMS
2	SEND_SMS
3	RECEIVE_SMS
4	WRITE_SMS
5	PROCESS_OUTGOING_CALLS
6	MOUNT_UNMOUNT_FILESYSTEMS
7	READ_HISTORY_BOOKMARKS
8	WRITE_HISTORY_BOOKMARKS
9	READ_LOGS
10	INSTALL_PACKAGES
11	MODIFY_PHONE_STATE
12	READ_PHONE_STATE
13	INTERNET
14	CALL_PHONE

(Danh sách 2) Intent-filter (action) nguy hiểm được liệt kê trong bảng 2.7.

Bảng 0.6: Các Intent-filter (action) nguy hiểm

Số thứ tự	Intent-filter (action)
1	BOOT_COMPLETED
2	SMS_RECEIVED
3	CONNECTIVITY_CHANGE
4	USER_PRESENT
5	PHONE_STATE
6	NEW_OUTGOING_CALL
7	UNINSTALL_SHORTCUT

8	INSTALL_SHORTCUT
9	left_up
10	right_up
11	left_down
12	right_down
13	SIG_STR
14	VIEW (chuỗi lành tính)

(Danh sách 3) Intent-filter (category) nguy hiểm được liệt kê trong bảng 2.8.

Bảng 0.7: Intent-filter (category) nguy hiểm

Số thứ tự	Intent-filter (category)
1	HOME
2	BROWSABLE (chuỗi lành tính)

(Danh sách 4) Tên Process nguy hiểm được liệt kê trong bảng 2.9.

Bảng 0.8: Tên Process nguy hiểm

Số thứ tự	Process
1	remote2
2	main
3	two
4	three

Sau khi thực hiện thu thập được danh sách từ khóa(keyword), ta sẽ thực hiện tính điểm số ác tính cho các thông tin thu được từ tệp tin manifest. Đầu tiên ta cần thực hiện phân loại các thuộc tính thu được từ tệp tin manifest là lành tính hay độc hại. Sau đó áp dụng công thức sau để tính [16]:

$$P = \frac{M - B}{E}$$

Trong đó:

P: điểm số ác tính

M: số lượng chuỗi độc hại

B: số chuỗi lành tính

E: tổng số thông tin thu được

Ví dụ như sau:

Trong tệp tin manifest ta thu được các quyền như sau:

<uses-permission
android:name="android.permission.READ_PHONE_STATE"/>

<uses-permission android:name="android.permission.READ_CONTACTS"/>

<uses-permission android:name="android.permission.WRITE_CONTACTS"/>

<uses-permission android:name="android.permission.INTERNET"/>

<uses-permission android:name="android.permission.CALL_PHONE"/>

<uses-permission android:name="android.permission.GET_TASKS"/>

<uses-permission android:name="android.permission.READ_LOGS"/>

Ta có tất cả là 7 quyền trong đó INTERNET, CALL_PHONE, READ_LOGS và READ_PHONE_STATE có trong danh sách các quyền nguy hiểm.

Vì vậy điểm số ác tính của mẫu được tính theo công thức:

$$P = \frac{4-0}{7} = 0.57$$

Thực hiện đánh giá tương tự với các thuộc tính còn lại gồm (2) Intent filter (action), (3) Intent filter (category), và (4) Process name.

Phương pháp đề xuất cung cấp các giá trị ngưỡng cho điểm số ác tính. Để có giá trị ngưỡng chính xác ta cần áp dụng các Kỹ thuật thống kê trung bình và trong phương pháp với tập dữ liệu đủ lớn để có được giá trị ngưỡng tốt nhất cho điểm số ác tính. Bằng cách sử dụng bốn thông tin thu được từ (1), (2), (3) và (4). Trong nghiên cứu này dựa vào việc phân tích với 50 mẫu mã độc chúng tôi đề xuất giá trị ngưỡng là 0.48.

3.1.3. Đánh giá và kết luận

Kết luận cuối cùng dựa trên cơ sở của điều kiện 1, 2 và công thức được đưa dưới đây. Điều kiện 1 mô tả đặc tính của phần mềm độc hại. Điều kiện hai được

thực hiện để tránh các kết luận không chính xác. Trong công thức dưới đây, SCORE được đề cập là điểm số ác tính cuối cùng của mẫu. C1 và C2 đếm số lượng hài lòng của một mẫu ở điều kiện 1 và 2 tương ứng.

Điều kiện 1:

- Điểm số ác tính lớn hơn giá trị ngưỡng.
- Số lượng Intent filter (priority) lớn hơn giá trị ngưỡng

Điều kiện 2:

- Điểm của Intent-filter (action) là số âm (< 0)
- Điểm của Intent-filter (category) là số âm (< 0)

Công thức:

SCORE = C1 – C2

Nếu SCORE cuối cùng lớn hơn hoặc bằng 1, thì mẫu thử nghiệm được coi là phần mềm độc hại.

Ưu điểm và nhược điểm của phương pháp

Ưu điểm:

Ưu điểm của phương pháp này là nó chỉ sử dụng các tệp tin manifest để phát hiện phần mềm độc hại. Tệp tin manifest có trong tất cả các ứng dụng Android vì vậy phương pháp này có thể áp dụng cho cả các phần mềm độc hại chưa được biết đến mà không thể phát hiện thông qua phương pháp phát hiện bằng chữ kí thông thường. Hơn nữa việc chỉ phân tích tệp tin manifest sẽ làm cho chi phí phân tích thấp hơn.

Phương pháp được này có thể phân loại chính xác các ứng dụng Android. Với dữ liệu càng lớn thì độ chính xác của phương pháp càng cao. Độ chính xác sẽ phụ thuộc vào giá trị ngưỡng được đề xuất.

Nhược điểm:

Một số mẫu mã độc không bị phát hiện bởi phương pháp này. Phương pháp này vẫn chưa đủ để phát hiện ra các phần mềm quảng cáo. Ngoài hiển thị các thông tin quảng cáo không cần thiết, đồng thời nó chứa một số sự khác biệt nằm ở biên của ứng dụng lành tính và ứng dụng quảng cáo. Điều này gây khó khăn cho phương pháp này trong quá trình phân loại các đặc tính đó.

3.2. Hạn chế nguy cơ mất an toàn

Có rất nhiều biện pháp có thể được sử dụng nhằm giản thiểu các cuộc tấn công của các phần mềm độc hại trên Android.

Bảo mật ngay từ phía nhà phát triển: các nhà phát triển luôn phải đặt việc bảo vệ dữ liệu cho người dùng lên hàng đầu. Như ta đã biết mọi ứng dụng đều được gán UID cho riêng mình. Tuy nhiên nó có thể cho bất kì chương trình nào yêu cầu chạy với UID của ứng dụng đó. Vì lí do này mà mọi ứng dụng cần phải có chữ kí của nhà phát triển. Trong Android, việc kí lên các ứng dụng thường được thực hiện bằng các chứng chỉ tự kí. Lý do chính để khẳng định về ký mã là cho phép các nhà phát triển để thực hiện cập nhật cho các ứng dụng hiện có của họ. Các ứng dụng khác nhau được ký sử dụng cùng khóa có thể yêu cầu để chạy với cùng một ID khi chúng được phát triển bởi cùng một nhà phát triển. Do đó cần phải lưu ý tới chữ kí của ứng dụng. Do đó các nhà phát triển có thể không được tin cậy một cách dễ dàng do đó Android nên cảnh báo và quy định người ký tên đáng tin cậy để bảo vệ an toàn cho dữ liệu người dùng [11].

Trách nhiệm từ phía người dùng: Một người sử dụng khi cài đặt các ứng dụng cần phải cấp quyền truy cập cho các quyền được yêu cầu khi cài đặt. Do đó, rất quan trọng đối với một người sử dụng là phải hiểu rõ các điều khoản theo yêu cầu của một ứng dụng. Một số ứng dụng như Skype mà cần nhiều quyền truy cập vào dữ liệu khác nhau trên điện thoại: như danh bạ, ảnh, vị trí, đọc thẻ nhớ,... Do đó người dùng cũng cần phải biết được các quyền mà ứng dụng yêu cầu có là bất thường hay không để quyết định có nên tiếp tục cài đặt phần mềm hay không [11].

Bảng 0.9: Một số quyền cơ bản và mức độ nguy hiểm đối với hệ thống [8]

STT	Quyền	Mức độ rủi ro	Tóm lược
1	Directly call numbers (Quay số trực tiếp)	Cao - Vừa phải	Quyền này cho phép ứng dụng gọi điện trực tiếp từ máy của người dùng. Nó có thể dẫn tới

			tiêu tốn tiền.
2	Send SMS (Gửi tin nhắn)	Cao - Vừa phải	Cho phép ứng dụng gửi tin nhắn văn bản đại diện cho người dùng.
3	Delete/Modify SD card contents. (Xóa/ sửa đổi nội dung trên thẻ nhớ ngoài)	Cao - Vừa phải	Điều này sẽ cho phép một ứng dụng để đọc, viết hoặc xóa dữ liệu được lưu trữ trên thẻ nhớ ngoài. Các dữ liệu có thể là hình ảnh, video, hoặc dữ liệu được ghi bởi các ứng dụng khác
4	Read Phone State and ID (Đọc thông tin điện thoại và định danh của điện thoại)	Vừa phải	Đây là yêu cầu của các ứng dụng như trò chơi để tạm dừng hoặc thực hiện một số nhiệm vụ khi người dùng nhận được một cuộc gọi. Tuy nhiên, cũng có thể là các phần mềm độc hại yêu cầu để lấy các thông tin về UID, IMEI, IMSI. Các thông tin có thể được gửi cho kẻ tấn công để lấy thông tin về thiết bị.
5	Read Contact Data (Đọc dữ liệu về danh bạ)	Vừa phải	Ứng dụng có thể đọc các số liên lạc có trong danh bạ người dùng sau đó thực hiện gửi về cho kẻ tấn công nhằm phát tán tin nhắn rác hoặc các phần mềm, mã độc hại khác.
6	Full Internet Access (Toàn	Cao - Vừa phải	Đây là một trong những quyền phổ biến nhất và nguy hiểm. Sự

	quyền thiết lập kết nối internet)		cho phép này được yêu cầu của tất cả các ứng dụng có hỗ trợ quảng cáo, trò chơi,...
7	Find GPS Location	Cao - Vừa phải	Sự cho phép này có thể cho phép một ứng dụng để theo dõi vị trí của người sử dụng và thu thập thông tin liên quan đến người sử dụng.

Dưới đây là một số thống kê các quyền được sử dụng của các malware [2]:

Bảng 0.10: Top 10 các Permission nguy hiểm mà các Spy sử dụng

Số thứ tự	Tên Permission	Số lần xuất hiện
1	READ_PHONE_STATE	845
2	SEND_SMS	802
3	INTERNET	799
4	RECEIVE_SMS	757
5	WRITE_EXTERNAL_STORAGE	732
6	READ_SMS	712
7	CALL_PHONE	701
8	READ_CONTACTS	501
9	CHANGE_WIFI_STATE	432
10	WRITE_SMS	387

Bảng 0.11: Top 10 các Permission nguy hiểm mà các Ransomware sử dụng

Số thứ tự	Tên Permission	Số lần xuất hiện
1	READ_PHONE_STATE	585
2	INTERNET	581
3	WRITE_EXTERNAL_STORAGE	201
4	GET_TASKS	165
5	CAMERA	128
6	RECEIVE_SMS	114
7	SYSTEM_ALERT_WINDOW	109
8	READ_CONTACTS	102
9	BATTERT_STATS	73
10	SEND_SMS	69

Bảng 0.12: Top 10 các Permission nguy hiểm mà các Trojan sử dụng

Số thứ tự	Tên Permission	Số lần xuất hiện
1	INTERNET	3528
2	READ_PHONE_STATE	3245
3	SEND_SMS	2016
4	WRITE_EXTERNAL_STORAGE	1513
5	RECEIVE_SMS	1498
6	WRITE_SMS	1436
7	ACCESS_FINE_LOCATION	1247
8	ACCESS_COARSE_LOCATION	982

9	GET_TASKS	899
10	READ_SMS	631

Cài đặt phần mềm diệt virus: cài đặt phần mềm diệt virus trên điện thoại di động, giống như trên máy tính để bàn, sẽ giúp xác định các malwares và ngăn chặn người dùng từ việc cài đặt chúng. Có một số phần mềm antivirus có sẵn trên điện thoại Android giúp làm sạch các malware và bảo vệ dữ liệu người dùng [13].

Ngoài ra, ta cần lưu ý một số vấn đề sau [17]:

- Chỉ nên cài đặt phần mềm từ các nguồn tin cậy như Google Play Store. Khi cài đặt cần chú ý tới mức độ đánh giá của người dùng khác với ứng dụng đó, tên nhà phát triển ứng dụng.
- Khi yêu cần đăng nhập, phải cá minh địa chỉ trang web là phù hợp và được hỗ trợ bởi ứng dụng.
- Một mật khẩu nên được thiết lập trên thiết bị trong trường hợp trộm cắp để làm phức tạp hơn quá trình truy cập dữ liệu trên thiết bị.
- Cần để ý tới các hiện tượng bất thường trên thiết bị như pin sụt bất thường, màn hình tự động sáng, hay tài khoản bị trừ bất thường,..

KẾT LUẬN

Ba chương của tiểu luận đã thể hiện được rằng những mục tiêu đặt ra khi thực hiện báo cáo đều đã đạt được. Cụ thể:

Chương 1 đã hệ thống lại những kiến thức tổng quan hệ điều hành Android. Kiến trúc của hiện điều hành Android, cấu trúc tệp tin của hệ điều hành Android và mô hình bảo mật trong hệ điều hành Android

Chương 2 thực hiện nghiên cứu các Kỹ thuật dịch ngược, công cụ và các Kỹ thuật phân tích mã độc cho ứng dụng Android. Ngoài ra, tôi còn nêu ra các loại mã độc và các biện pháp giảm thiểu nguy cơ mất an toàn từ mã độc trên Android.

Trong chương 3 tôi có thực nghiệm phân tích mã độc trên ứng dụng Android bằng phương pháp phân tích tệp tin manifest. Tệp tin manifest có trong tất cả các ứng dụng Android vì vậy phương pháp này có thể áp dụng cho cả các phần mềm độc hại chưa được biết đến mà không thể phát hiện thông qua phương pháp phát hiện bằng chữ ký thông thường. Hơn nữa, việc chỉ phân tích tệp tin manifest sẽ làm cho chi phí phân tích thấp hơn. Đồng thời, phương pháp này cũng có thể kết hợp với các phương pháp khác để phát hiện chính xác hơn.

TÀI LIỆU THAM KHẢO

[1] Christian Lueg, “8,400 new Android malware samples every day”, G DATA Security Blog, 2017.

[2] Eric Chin, “Motivations of Recent Android Malware”, Symantec Security Response, Tech. Rep, 2011.

[3] Himanshu Shewale, Sameer Patil, Vaibhav Deshmukh and Pragya Singh, “Analysis of Android Vulnerabilities and Modern Exploitation Techniques”, in ICTACT Journal on Communication Technology, vol.5, no.1, 2014.

[4] Kindsight, “The Mobile Malware Problem”, in A Kindsight White Paper, Ottawa, Canada, Tech.Report, 2012.

[5] Muhammad Zuhair Qadir, Atif Nisar Jilani and Hassan Ullah Sheikh, “Automatic Feature Extraction, Categorization and Detection of Malicious Code in Android Application”, in Proceeding International Journal of Information and Network Security, vol.3, no.1, pp.12-17, 2014.

[6] Stefan Brahler, “Analysis of the Android Architecture”, Karlsruhe Institute of Technology, Tech. Rep, 2010.

[7] Justin Sahs and Latifur Khan, “A Machine Learning Approach to Android Malware Detection”, in Intelligence and Security Informatics Conference, Odense, European, 2012.

[8] Luoshi Zhang, Yan Niu, Xiao Wu, Zhaoguo Wang and Yibo Xue, “A3: Automatic Analysis of Android Malware”, in International Workshop on Cloud Computing and Information Security, 2013.

[9] Alessandro Armando, Alessio Merlo and Luca Verderama, “Security Issues in the Android crosslayer architecture”, 2012.

[10] Kevin Allix, Tegawende Bissyande, Quentin Jerome, Jacques Klein and Radu State, “LargeScale Machine Learning-based Malware Detection: Confronting the “10-Fold Cross Validation” Scheme with Reality”, in Conference on Data and Application Security and Privacy, San Antonio, Texas, USA, 2014.

[11] Zami Aung and Win Zaw, “Permission-Based Android Malware Detection”, in International Journal of Scientific & Technology Research, vol.2, no.3, 2013.

[12] Yousra Aafer, Wenliang Du and Heng Yin, “DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android”, in Security and Privacy in Communication Networks, pp. 86-103, 2013.

[13] Detecting Android Malware by Analyzing Manifest Files: Ryo Sato1, Daiki Chiba and Shigeki Goto.

[14] Nguyễn Minh Đức, “Phân tích mã độc trên Android và dự đoán xu hướng năm 2015”, SecurityDaily, 2015.

[15] Troy Vennon, GTC Research Engineer, “A Study of Known and PotentialMalware Threats”, 2010. Journal of Science and Technology on Information security 18 Số 1.CS (05) 2017

[16] Isohara T.; Kawabata H.; Yakemori K.; Kubota A.; Kani J.; Agematsu H.; Nishigaki A. Detection Technique of Android Malware with Second Application. Proceedings of Computer Security Symposium.

[17] Enck W.; Ongtang M.; McDaniel P. On Lightweight Mobile Phone Application Certification.

[18] Wu D.; Mao C.; Wei T.; Lee H.; Wu K. DroidMat: Android Malware Detection throug Manifest and API Calls Tracing. Seventh Asia Joint Conference on Information Security.

[19] https://play.google.com/store [20] https://virusshare.com

[21] http://www.cs.waikato.ac.nz/ml/weka/

[22] Lê Bá Cường; Trịnh Doãn Mạnh; “Phát hiện mã độc trên Android bằng phương pháp phân tích tập tin manifest” in Nghiên cứu Khoa học và Công nghệ trong lĩnh vực An toàn thông tin, 2017