

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ



ĐỒ ÁN TỐT NGHIỆP

**XÂY DỰNG HỆ THỐNG GIÁM SÁT HÀNH VI MÃ
ĐỘC TRÊN HỆ ĐIỀU HÀNH ANDROID**

Ngành: An toàn thông tin
Mã số: 7.48.02.02

Hà Nội, 2018

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ



ĐỒ ÁN TỐT NGHIỆP

**XÂY DỰNG HỆ THỐNG GIÁM SÁT HÀNH VI MÃ
ĐỘC TRÊN HỆ ĐIỀU HÀNH ANDROID**

Ngành: An toàn thông tin
Mã số: 7.48.02.02

Sinh viên thực hiện:
Dương Đình Tân
Lớp: AT10B

Người hướng dẫn:
PGS. TS. Lương Thế Dũng
Khoa An toàn thông tin – Học viện Kỹ thuật mật mã

Hà Nội, 2018

MỤC LỤC

Mục lục	i
Danh mục kí hiệu và viết tắt.....	iii
Danh mục hình vẽ.....	iv
Danh mục bảng.....	v
Lời cảm ơn	vi
Lời nói đầu	vii
Chương 1. Hệ điều hành Android và mã độc trên hệ điều hành Android.....	1
1.1. Hệ điều hành Android	1
1.2. Mô hình kiến trúc hệ điều hành android	3
1.3. Các cơ chế đảm bảo an toàn được triển khai trong Android	5
1.3.1. <i>Nền tảng và hỗ trợ từ Google</i>	5
1.3.2. <i>An toàn trong kernel linux</i>	10
1.3.3. <i>Bảo mật ứng dụng</i>	12
1.4. Mã độc trên Android	15
1.4.1. <i>Hiện trạng</i>	15
1.4.2. <i>File thực thi trên Android</i>	16
1.5. Dự án mã nguồn mở Android	17
1.5.1. <i>Quản lý mã nguồn AOSP</i>	18
1.5.2. <i>Tên mã, nhãn và số hiệu bản dựng</i>	19
Chương 2. Thiết kế và xây dựng hệ thống giám sát hành vi mã độc trên hệ điều hành Android	21
2.1. Thiết kế hệ thống.....	21
2.1.1. <i>Nguyên lý làm việc</i>	21
2.1.2. <i>Mô hình hệ thống</i>	22
2.1.3. <i>Tương tác giữa các thành phần trong hệ thống</i>	22
2.1.4. <i>Chi tiết hoạt động của hệ thống phân tích</i>	23
2.1.5. <i>Các công nghệ sử dụng</i>	25

2.2. Xây dựng hệ thống	26
2.2.1. <i>Xây dựng sandbox phân tích</i>	26
2.2.2. <i>Xây dựng thành điều khiển máy ảo Android</i>	37
2.2.3. <i>Xây dựng thành phần trang web quản trị</i>	41
Chương 3. Thử nghiệm hệ thống và đánh giá kết quả	45
3.1. Mô hình thử nghiệm.....	45
3.1.1. <i>Các thành phần</i>	45
3.1.2. <i>Mẫu mã độc sử dụng</i>	45
3.2. Phân tích kết quả phân tích của hệ thống.....	45
3.3. Đánh giá kết quả phân tích.....	48
3.3.1. <i>So sánh với kết quả phân tích tĩnh</i>	48
3.3.2. <i>Độ chính xác của hệ thống</i>	49
Kết Luận.....	51
Phụ lục:.....	52
Tài liệu tham khảo.....	56

DANH MỤC KÍ HIỆU VÀ VIẾT TẮT

Từ viết tắt	Tiếng Anh	Tiếng việt
ADB	Android debug bridge	Cầu nối gỡ lỗi android
AOSP	Android open source project	Dự án mã nguồn mở Android
ART	Android runtime	Trình thực thi Android
API	Application programming interface	Giao diện lập trình ứng dụng
C&C	Command and Control	Gia lệnh và điều khiển
JVM	Java virtual machine	Máy ảo java
OTA	Over the air	Cập nhật không dây

DANH MỤC HÌNH VẼ

Hình 1.1 Mô hình kiến trúc Android.....	3
Hình 1.2 Các thành phần trong Android	7
Hình 1.3 Cấu trúc tệp tin apk	16
Hình 1.4 Quá trình biên dịch.....	17
Hình 1.5 Các nhánh phát hành AOSP.....	18
Hình 2.1 Không gian hoạt động của Sandbox	21
Hình 2.2 Mô hình hệ thống	22
Hình 2.3 Tương tác với người dùng.....	23
Hình 2.4 Luồng hoạt động của hệ thống.....	25
Hình 2.5 Sửa số IMEI	36
Hình 2.6 Sửa số điện thoại	36
Hình 3.1 Các kết nối của mã độc	46
Hình 3.2 Mã độc thao tác với file dex bổ sung	47
Hình 3.3 File dex được thực thi	47
Hình 3.4 File blue.png được trích xuất	48
Hình 3.5 Một số chuỗi sau khi được giải mã	49

DANH MỤC BẢNG

Bảng 1.1 Tên mã và ngày phát hành các phiên bản Android	2
Bảng 1.2 Phiên bản kernel linux đi kèm AOSP	3
Bảng 1.3 Mức API theo phiên bản AOSP.....	19
Bảng 1.4 Nhãn các nhánh AOSP	20

LỜI CẢM ƠN

Trong quá trình thực hiện đồ án tốt nghiệp này, tôi đã nhận được sự giúp đỡ tận tình của thầy giáo hướng dẫn là PGS. TS. Lương Thế Dũng – Giảng viên Khoa An toàn thông tin Học viện Kỹ thuật Mật mã, sự động viên của người thân và bạn bè.

Xin cảm ơn tất cả mọi người đã tạo những điều kiện tốt nhất để tôi hoàn thành đồ án tốt nghiệp này!

SINH VIÊN THỰC HIỆN ĐỒ ÁN

Dương Đình Tân

LỜI NÓI ĐẦU

Trong quá trình làm việc và nghiên cứu về mã độc trên hệ điều hành Android, việc sử dụng một sandbox để thực thi và phân tích mã độc là một bước giúp thu thập nhiều thông tin hữu ích, định hướng cho việc phân tích mã thực thi của mã độc. Các hệ thống phân tích hiện tại cung cấp các tính năng khá nghèo nàn về tính năng, cùng với tính đóng của sản phẩm. Trong khi đó các hệ thống phân tích dựa trên mã nguồn mở lại có nhiều vấn đề như chưa hoàn thiện, cung cấp chưa đầy đủ thông tin, hoạt động không ổn định, dễ bị phát hiện nếu người viết mã độc có kinh nghiệm trong việc kiểm tra môi trường thực thi và vượt qua dễ dàng.

Từ những lý do trên, việc nghiên cứu xây dựng hệ thống giám sát hành vi, phân tích động là cần thiết và mang lại nhiều giá trị.

Đề tài nghiên cứu xây dựng hệ thống phân tích động dựa trên dự án mã nguồn mở hệ điều hành Android (Android open source project – AOSP), cũng như mục tiêu cần đạt được của đồ án tốt nghiệp này là:

1. Tùý bién lại bộ API, nhằm mục đích giám sát hành vi mã độc.
2. Xây dựng hệ thống lưu trữ kết quả.
3. Xây dựng hệ thống tương tác với người dùng để đẩy mẫu lên phân tích và kiểm tra kết quả.

SINH VIÊN THỰC HIỆN ĐỒ ÁN

Dương Đình Tân

CHƯƠNG 1. HỆ ĐIỀU HÀNH ANDROID VÀ MÃ ĐỘC TRÊN HỆ ĐIỀU HÀNH ANDROID

1.1. Hệ điều hành Android

Hệ điều hành android là một bộ mã nguồn mở được tạo ra cho một diện rộng các thiết bị khác nhau, từ các nhà sản xuất phần cứng khác nhau. Mục đích chính của android là tạo nên một nền tảng mở cho các nhà sản xuất, các nhà phát triển phần mềm, để mang các ý tưởng vào thực tế, trong thực tế những sản phẩm đó làm tăng trải nghiệm di động cho người dùng.

Họ (đội phát triển android) cũng muốn đảm bảo tính cạnh tranh giữa các nhà phát triển, không ai có thể cản trở tính cạnh tranh, hay kiểm soát sự phát triển của các nhà phát triển khác.

Android được dẫn đầu bởi nhóm các công ty, được biết đến như liên minh thiết bị cầm tay (Open Handset Alliance - OHA) được dẫn đầu bởi Google. Ngày nay, rất nhiều các công ty, bao gồm cả các thành viên chính thức của OHA và các công ty khác đang đầu tư rất nhiều vào Android. Các công ty này đầu tư các nhân tố kỹ thuật quan trọng nhằm cải thiện android và mang các sản phẩm chạy android ra thị trường.

Các công ty đã đầu tư vào android vì họ tin vào giá trị và sự cần thiết của một nền tảng mở cho tất cả. Android là nền tảng mở hoàn toàn và có chủ ý. Các nhóm tổ chức chia sẻ tài nguyên cần thiết cho việc phát triển giữa các bên đơn lẻ khi triển khai các sản phẩm.

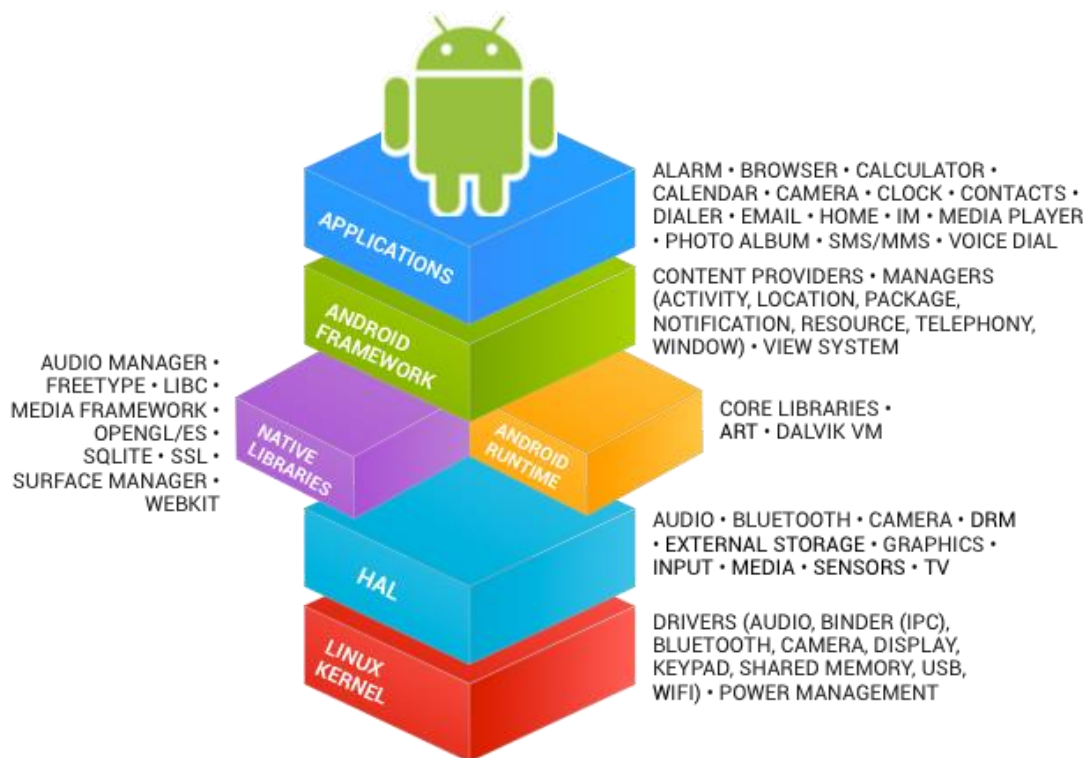
Dự án mã nguồn mở android được dẫn đầu bởi Google, người duy trì mà phát triển. mặc dù Android bao gồm nhiều dự án nhỏ, được quản lý chặt chẽ. Họ xem việc quản lý android như một việc riêng biệt, một sản phẩm phần mềm toàn diện, không phải một bản phân phối, một sự đặc biệt, hay một bộ các phần có thể thay thế được.

Về lịch sử, Android được google chính thức ra mắt phiên bản đầu tiên vào năm 2008, tuy nhiên thời điểm này lại không được cộng đồng chú ý. Tên mã và ngày phát hành được thể hiện ở bảng 1.1

Bảng 1.1 Tên mã và ngày phát hành các phiên bản Android

Tên mã	Số phiên bản	Ngày phát hành	Mức API
No codename	1.0	23/09/2008	1
Internally known as "Petit Four"	1.1	09/01/2009	2
Cupcake	1.5	27/04/2009	3
Donut	1.6	15/09/2009	4
Eclair	2.0 – 2.1	26/10/2009	5 – 7
Froyo	2.2 – 2.2.3	20/05/ 2010	8
Gingerbread	2.3 – 2.3.7	06/12/2010	9 – 10
Honeycomb	3.0 – 3.2.6	22/02/2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.4	18/10/2011	14 – 15
Jelly Bean	4.1 – 4.3.1	09/07/2012	16 – 18
KitKat	4.4 – 4.4.4	31/10/2013	19 – 20
Lollipop	5.0 – 5.1.1	12/11/2014	21 – 22
Marshmallow	6.0 – 6.0.1	05/10/2015	23
Nougat	7.0 – 7.1.2	22/08/2016	24 – 25
Oreo	8.0 – 8.1	21/08/2017	26 – 27

1.2. Mô hình kiến trúc hệ điều hành android



Hình 1.1 Mô hình kiến trúc Android

Android được xây dựng trên nền tảng Linux, khi Android phát triển lên các phiên bản mới, bản thân nó cũng cập nhật cả phần nhân Linux, cụ thể với các phiên bản trong bảng 1.2 dưới đây.

Bảng 1.2 Phiên bản kernel linux đi kèm AOSP

Phiên bản	Tên mã	Mức API	Phiên bản nhân Linux
1.5	Cupcake	3	2.6.27
1.6	Donut	4	2.6.29
2.0/1	Eclair	5-7	2.6.29
2.2.x	Froyo	8	2.6.32
2.3.x	Gingerbread	9, 10	2.6.35
3.x.x	Honeycomb	11-13	2.6.36
4.0.x	Ice Cream San	14, 15	3.0.1
4.1.x	Jelly Bean	16	3.0.31
4.2.x	Jelly Bean	17	3.4.0
4.3	Jelly Bean	18	3.4.39
4.4	Kit Kat	19, 20	3.10

5.x	Lollipop	21, 22	3.16.1
6.0	Marshmallow	23	3.18.10
7.0	Nougat	24	4.4.1
7.1	Nougat	25	4.4.1
8.0	Oreo	26	4.10

Phần nhân linux thực hiện các công việc chủ yếu về Driver (âm thanh, kết nối, camera, màn hình, bàn phím ...) và quản lý năng lượng.

Phần HAL (Hardware abstract layer) lớp phần cứng ảo, phần này tạo ra các thiết bị phần cứng ảo, bao gồm âm thanh, hình ảnh, camera, bộ nhớ lưu trữ, đồ họa, các cảm biến. Việc tạo ra các lớp ảo cho phép Android phân chia thành từng lớp, tách biệt các lớp thuận tiện cho việc phát triển các phiên bản mới và đặt biệt do Android phát triển cho một lượng lớn các thiết bị phần cứng khác nhau, bao gồm từ các thiết bị di động đến từ các nhà phát triển phần cứng khác nhau, máy tính bảng, TV hay các thiết bị thông minh khác.

Phần tiếp theo, bao gồm các thư viện thuần (native libraries) và android runtime. Thư viện thuần bao gồm các thư viện xử lý các tác vụ nền tảng như: quản lý âm thanh, thư viện đồ họa (OpenGL), thư viện quản lý dữ liệu(SQLite), thư viện mã hóa SSL, DES, AES ... và Webkit. Phần android runtime bao gồm các thư viện cốt lõi của hệ thống và máy ảo android(Android runtime – ART), máy ảo Dalvik.

Phần Android framework cung cấp các thành phần, môi trường các dịch vụ cho tầng ứng dụng phía trên thực thi bao gồm: contentprovider, manager(activity, location, package, notification, resource, telephony, window) và ViewSystem.

Tầng trên cùng, đây là tầng thực thi của các ứng dụng được người dùng tương tác, các ứng dụng này bao gồm cả các ứng dụng được nhà sản xuất cài sẵn và các ứng dụng được người dùng cài thêm từ nhà phát triển thứ ba như tin nhắn, trình duyệt, báo thức, máy ảnh, đồng hồ ...

1.3. Các cơ chế đảm bảo an toàn được triển khai trong Android

1.3.1. Nền tảng và hỗ trợ từ Google

1.3.1.1. Tổng quan

Android kết hợp các nghiên cứu về tính năng bảo mật hàng đầu, và làm việc với các nhà phát triển cùng với các nhà sản xuất thiết bị để giữ cho nền tảng android và hệ sinh thái android được an toàn. Một mô hình bảo mật mạnh mẽ được sử dụng để thiết lập và cho phép tạo nên một nền tảng an toàn cho các ứng dụng và các thiết bị được xây dựng xoay quanh nền tảng android với sự trợ giúp của các dịch vụ điện toán đám mây. Kết quả là, xuyên suốt vòng đời phát triển, Android trở thành nền tảng với phần được đảm bảo an toàn.

Android được thiết kế để trở nên “mở”. Ứng dụng Android sử dụng các các phần mềm và phần cứng tiên tiến, cũng như các dữ liệu cục bộ và dữ liệu trên máy chủ cho thấy khả năng phát triển và giá trị cho khách hàng. Để làm việc này, nền tảng cung cấp môi trường cho ứng dụng thực thi, đảm bảo tính bí mật, toàn vẹn và sẵn sàng cho dữ liệu, ứng dụng, thiết bị và kết nối.

An toàn trong một nền tảng mở yêu cầu một kiến trúc an toàn vững chắc và các trường trình bảo mật mạnh mẽ. Android được thiết kế với việc đảm bảo an toàn qua nhiều lớp, việc này đủ phức tạp và linh hoạt trong việc bảo vệ tất cả người dùng sử dụng nền tảng.

Android được thiết kế cho những nhà phát triển. Các cơ chế kiểm soát an toàn được thiết kế nhằm giảm gánh nặng cho các nhà phát triển. Khi hiểu về cách kiểm soát an toàn, người phát triển có thể dễ dàng làm việc với hệ thống, tuy nhiên họ cũng không cần biết quá nhiều về bảo mật, vẫn có thể bảo vệ dữ liệu của họ theo cách mặc định của hệ thống.

Cùng với việc cung cấp các cơ chế trên. Android cũng cung cấp thêm các trợ giúp cho nhà phát triển theo một số cách khác. Đội ngũ bảo mật có thể tìm ra các nguy cơ về bảo mật trong ứng dụng và đưa ra các giúp đỡ để họ khắc phục vấn đề.

Đối với các thiết bị có Google Play (kho ứng dụng của google), Play Service cung cấp các bản cập nhật cho các thư viện phần mềm như OpenSSL, thứ được dùng cho các ứng dụng truyền thông an toàn.

Android được thiết kế cho người dùng. Người dùng Android được cung cấp khả năng có thể nhận ra các quyền được ứng dụng yêu cầu cho từng ứng dụng và kiểm soát qua các quyền đó. Kiểu thiết kế này cho phép chống lại một số kẻ tấn công đang thực hiện một số các tấn công phổ biến, như tấn công thiết bị người dùng bằng cách cài các phần mềm độc hại. Android được thiết kế để giảm khả năng bị tấn công cũng như giảm thiểu thiệt hại khi hệ thống bị tấn công thành công. Android vẫn tiếp tục trong quá trình nâng cao khả năng bảo mật khi người dùng tiếp tục sử dụng thiết bị sau này, thông qua các bản cập nhật được phối hợp cùng với các hãng sản xuất thiết bị tới người dùng cuối.

1.3.1.2. Nền tảng

Android được phát triển dựa trên mã nguồn mở, cùng với môi trường ứng dụng cho các thiết bị di động. Nội dung dưới đây mô tả tính năng bảo mật trong nền tảng Android, hình 1 mô tả các thành phần bảo mật và mục đích của các thành phần khác nhau trong nền tảng android. Từng thành phần đảm bảo cho các thành phần bên dưới được an toàn. Ngoại lệ chỉ số ít các mã của hệ điều hành android được chạy dưới quyền root, tất cả các mã phía trên kernel linux đều được giới hạn truy cập thông qua android sandbox. Các thành phần trong Android được mô tả trong hình 1.2



Hình 1.2 Các thành phần trong Android

Các thành phần chính của Android được xây dựng trên các khối sau:

- Thiết bị phần cứng: Android chạy trên một diện rộng các phần cứng khác nhau, bao gồm từ điện thoại, máy tính bảng, đồng hồ, tv, các máy chơi game. Android hoạt động theo cách giống nhau trên các nền tảng vi xử lý khác nhau. Nhưng có thể tận dụng được 1 số tính năng nâng cao của phần cứng như ARM eXecute-Never (ARM NX)
- Hệ thống android: phần lõi hệ thống Android được xây dựng trên nền tảng linux, tất cả các tài nguyên thiết bị như chức năng chụp ảnh, dữ liệu GPS, bluetooth, gọi điện, kết nối mạng đều được truy cập thông qua hệ điều hành.
- Ứng dụng thực thi: ứng dụng Android hầu hết được viết bằng ngôn ngữ lập trình java, và chạy trên Android runtime (ART). tuy nhiên một số ứng dụng bao gồm cả một số dịch vụ cốt lõi trong Android đều là các ứng dụng native hoặc sử dụng các thư viện native. Cả ART và các ứng dụng native chạy trong

cùng một môi trường bảo mật như nhau. Các ứng dụng được hệ thống dành riêng một không gian trong hệ thống tập tin, nơi mà nó có toàn quyền đọc ghi dữ liệu, bao gồm cả các file dữ liệu.

Ứng dụng Android mở rộng cho các ứng dụng có trên hệ thống. Có hai loại ứng dụng chính trên Android.

- Ứng dụng được cài trước: Android bao gồm một số ứng dụng được cài từ trước bao gồm: điện thoại, email, lịch, trình duyệt, danh bạ. các ứng dụng này có thể là một phần của AOSP.
- Ứng dụng do người dùng cài: Android cung cấp môi trường mở, hỗ trợ các bên thứ ba phát triển ứng dụng. Google Play cung cấp cho người dùng hàng trăm nghìn các ứng dụng như vậy.

1.3.1.3. Dịch vụ bảo mật của google

- Google play: Google cung cấp những dịch vụ cho phép người dùng khám phá, cài đặt thanh toán trong ứng dụng từ những ứng dụng của họ. Google play cho phép các nhà phát triển dễ dàng hơn trong việc tương tác với người dùng.
- Cập nhật android: Android cung cấp khả năng cập nhật tới các thiết bị, cho phép cập nhật các tính năng bảo vệ người dùng thông qua các trang web hoặc OTA.
- Xác thực ứng dụng: cảnh báo hoặc tự động chặn việc cài đặt các ứng dụng độc hại, tiếp tục quét các ứng dụng đã có trên thiết bị, đưa ra cảnh báo hoặc gỡ bỏ ứng dụng độc hại.

1.3.1.4. Tổng quan về bảo mật trương trình

Điểm mấu chốt của trương trình bảo mật Android bao gồm:

- Kiểm tra thiết kế: quá trình phát triển bảo vệ cho android được bắt đầu rất sớm trong quá trình xây dựng mô hình an toàn. Mỗi một tính năng của nền tảng được xem xét bởi các kỹ sư và các nhà nghiêm cứu bảo mật, với các tiếp cận đảm bảo kiểm soát hoàn toàn kiến trúc của hệ thống.

- Kiểm thử và rà soát mã nguồn: Trong suốt quá trình phát triển nền tảng, các thành phần mã nguồn mở là đối tượng để kiểm soát chặt chẽ mã nguồn. việc rà soát mã nguồn được thực hiện bởi đội ngũ bảo mật của Android, các kỹ sư an ninh mạng của Google, cùng với các chuyên gia an ninh mạng làm việc tự do khác. Mục tiêu của việc rà soát mã nguồn là xác định các điểm yếu, những điểm có thể có lỗ hổng bảo mật, trước khi được phát hành.
- Mã nguồn mở và góp sức của cộng đồng: dự án mã nguồn mở Android cho phép bất cứ ai có thể thực hiện việc rà soát mã nguồn. Android cũng khuyến khích thực hiện rà soát với các thành phần mã nguồn mở của các bên khác, ví dụ như phần nhân Linux. Google play cung cấp diễn đàn cho chép người dùng và các công ty trao đổi thông tin với nhau về các ứng dụng đặc biệt nào đó.
- Khắc phục sự cố: ngay cả khi thực hiện tất cả các biện pháp bảo vệ trên. Các vấn đề về bảo mật vẫn có thể xuất hiện, đó cũng là lý do tại sao Android đã tạo ra cả một quy trình toàn diện trong việc xử lý các sự cố về bảo mật. Đội ngũ bảo mật android dành thời gian để giám sát và theo dõi các cộng đồng về bảo mật, nơi các vấn đề bảo mật được thảo luận và lưu trữ lại các vấn về bảo mật của Android vào cơ sở dữ liệu. Ngoài ra việc bảo vệ người dùng từ xa bao gồm các việc như: cập nhật nền tảng, gỡ bỏ các phần mềm độc hại từ Google play cũng như các ứng dụng đang được cài trên máy người dùng.

1.3.1.5. Kiến trúc bảo mật của nền tảng

Android trở nên an toàn hơn và tiện dụng hơn cho các thiết bị di động bằng các tái định nghĩa bảo mật truyền thống thành:

- Bảo vệ dữ liệu ứng dụng và người dùng.
- Bảo vệ tài nguyên(bao gồm cả mạng)
- Cung cấp tính năng cô lập ứng dụng với hệ thống, với ứng dụng khác, với người dùng.

Để đạt được các mục tiêu trên, Android cung cấp các tính năng bảo mật mẫu chốt sau:

- Bảo mật mạnh mẽ hệ điều hành thông qua nhân linux
- Triển khai sandbox bắt buộc cho tất cả các ứng dụng
- Giao tiếp tiến trình an toàn
- Ký ứng dụng
- ứng dụng định nghĩa quyền và người dùng trao quyền

1.3.2. An toàn trong kernel linux

Ở mức hệ điều hành, nền tảng Android cung cấp tính năng bảo mật của hạt nhân linux, cũng như bảo mật trong giao tiếp giữa các tiến trình (inter-process communication - IPC) cho phép giao tiếp an toàn giữa các ứng dụng khi chạy trong các tiến trình khác nhau. Các tính năng bảo mật ở mức hệ điều hành này đảm bảo các ứng dụng native code cũng bị ràng buộc bởi hệ thống Android.

1.3.2.1. An toàn Linux

Android được phát triển trên nền tảng Linux, Linux ngày càng phát triển rộng qua các năm, và được sử dụng cho hàng triệu môi trường nhạ cảm. Thông qua lịch sử liên tục được các nhà nghiên cứu, các kẻ tấn công, và được sửa lỗi bởi hàng ngàn các lập trình viên, Linux trở thành nền tảng an toàn vững trãi được tin tưởng bởi các công ty, các chuyên gia bảo mật.

Linux cung cấp các tính năng bảo mật chính bao gồm:

- Mô hình phân quyền dựa trên người dùng.
- Cô lập tiến trình
- Mở rộng cơ chế an toàn cho IPC
- Có thể loại bỏ các thành phần không cần thiết và tiềm ẩn nguy cơ bảo mật của nhân linux

Như các hệ điều hành đa người dùng khác, một tính năng căn bản khác của nhân linux đó là cô lập tài nguyên với bên ngoài, triết lý bảo mật trên linux khi thực hiện việc này là:

- Ngăn chặn người dùng A đọc dữ liệu của người dùng B
- Đảm bảo người dùng A không xâm phạm vào bộ nhớ của B
- Đảm bảo người dùng A không xâm phạm tài nguyên CPU của B
- Đảm bảo người dùng A không xâm phạm vào thiết bị của B

1.3.2.2. Cơ chế hộp cát trong Android

Nền tảng Android triển khai cơ chế bảo vệ dựa trên người dùng, điều này có nghĩa là nó định danh và cô lập tài nguyên giữa các ứng dụng. Hệ thống Android đăng ký một ID(UID) duy nhất cho mỗi ứng dụng và chạy có trong một tiến trình tách biệt, các tiếp cận này khác với các hệ điều hành khác (kể cả với nền tảng Linux truyền thống), khi mà các ứng dụng sẽ chạy với cùng một quyền như nhau. Việc này tạo nên cơ chế an toàn ở mức hệ điều hành. Kernel sẽ thực hiện an toàn giữa các ứng dụng và hệ thống ở mức độ các tiến trình thông qua dựa trên cơ sở nền tảng của Linux, như việc gán các ID người dùng và nhóm cho các ứng dụng. mặc định các ứng dụng không thể tương tác với các ứng dụng khác vì bị giới hạn truy cập bởi hệ điều hành. Nếu ứng dụng A cố làm gì đó độc hại như đọc dữ liệu của người dùng B hoặc thực hiện một cuộc gọi mà không có quyền, sau đó hệ điều hành sẽ chống lại việc này bởi người A không được cho phép sử dụng đặc quyền này. Hộp cát khá đơn giản, và có thể kiểm soát, được dựa trên kiểu UNIX cũ, cô lập tiến trình và tệp tin người dùng.

1.3.2.3. Phân vùng hệ thống và chế độ an toàn

Phân vùng hệ thống chứa nhân của Android cũng như các thư viện hệ thống, các ứng dụng đang chạy, nền tảng Android, và các ứng dụng. Đây là phân vùng được đặt ở chế độ chỉ đọc. Khi người dùng khởi động ở chế độ an toàn (safe mode), các ứng dụng bên thứ ba có thể chạy thử công bởi người chủ thiết bị mà không cần khởi chạy như mặc định.

1.3.2.4. Phân quyền hệ thống tệp tin

Trong môi trường UNIX, hệ thống phân quyền tệp tin đảm bảo người dùng không thể thay đổi hay đọc dữ liệu của người dùng khác. Trong trường hợp Android, các ứng dụng thực thi với ngữ cảnh là các người dùng, vì vậy nó không thể truy cập file của ứng dụng khác, trừ khi các nhà phát triển cho phép điều này.

1.3.2.5. Mã hóa

Android cung cấp một bộ các API thuật toán mã hóa cho các ứng dụng sử dụng, việc này bao gồm triển khai các thuật toán chuẩn được nhúng sẵn như AES, RSA, DES và SHA, thêm vào đó các API cung cấp thêm ở mức cao hơn như SSL và HTTPS.

Android 4.0 giới thiệu lớp KeyChain cho phép ứng dụng sử dụng hệ thống lưu trữ thông tin an toàn cho các khóa bí mật và các chứng chỉ.

1.3.3. Bảo mật ứng dụng

1.3.3.1. Các thành phần của ứng dụng android

Phần nhân của hệ điều hành dựa trên Linux. Các ứng dụng trên Android được viết hầu hết bằng ngôn ngữ Java và chạy trên máy ảo Dalvik. Tuy nhiên ứng dụng có thể được viết bằng native code. Các ứng dụng được cài đặt từ một file với phần mở rộng .apk.

Các khối chính của file ứng dụng trên Android bao gồm:

- **AndroidManifest.xml:** là file điều khiển, nói với hệ điều hành rằng nó sẽ làm gì với phần trên cùng của hệ thống Android (các màn hình hoạt động, dịch vụ, các trình nhận quảng bá ...) trong một ứng dụng. Nó cũng mô tả các quyền được yêu cầu.
- **Activities:** một Activity là một phần mã nguồn cho một nhiệm vụ mà người dùng sẽ tập trung làm việc với nó. Nó thường được sử dụng để hiển thị giao diện tới người dùng.
- **Service:** một service là một phần mã sẽ được thực thi dưới nền. Nó có thể sở hữu tiến trình riêng, hoặc ngữ cảnh tiến trình khác của ứng dụng. Một thành phần khác có thể kết nối (bind) tới service và gọi các phương thức của nó

thông qua gọi thủ tục từ xa. Lấy ví dụ, trình phát đa phương tiện, khi người dùng thoát khỏi giao diện phát nhạc thì nhạc vẫn được tiếp tục phát.

- **Broadcast Receiver:** một Broadcast Receiver là một đối tượng được sử dụng để bắt các sự kiện trong hệ thống. Một ứng dụng có thể đăng ký để nhận các sự kiện như pin yếu, tin nhắn.

1.3.3.2. Mô hình phân quyền của Android: truy cập các API được bảo vệ

Tất cả các ứng dụng android đều được chạy trong “hộp cát”. Mặc định, một ứng dụng có thể truy cập có hạn chế vào một khoảng tài nguyên của hệ thống. Hệ thống quản lý ứng dụng truy cập vào các tài nguyên đó, nếu sử dụng không đúng cách hoặc nguy hại có thể tác động đến trải nghiệm người dùng, kết nối mạng, hoặc dữ liệu trên thiết bị.

Các hạn chế này được thực hiện dưới nhiều hình thức khác nhau. Một vài có thể hạn chế bằng cách hạn chế truy cập vào các API nhạy cảm (ví dụ: không có API nào cho phép điều khiển thẻ SIM). Trong nhiều trường hợp, sự phân tách vai trò của các API đảm bảo việc này, cùng với sự cô lập bộ nhớ cho từng ứng dụng. Trong trường hợp khác, các API nhạy cảm được dự định sử dụng cho các ứng dụng đáng tin cậy, được bảo vệ thông qua cơ chế bảo mật quyền truy cập.

Các API nhạy cảm được bảo vệ bao gồm:

- Tính năng máy ảnh
- Dữ liệu vị trí
- Tính năng Bluetooth
- Tính năng gọi điện
- Tính năng tin nhắn
- Kết nối mạng

1.3.3.3. Giao tiếp giữa các tiến trình

Các tiến trình có thể giao tiếp bằng sử dụng các cơ chế truyền thông trong UNIX. Ví dụ như systemfile, socket, signals. Tuy nhiên, hệ thống quyền vẫn được áp dụng.

Android cung cấp thêm các cơ chế IPC khác:

- **Binder:** một cơ chế gọn nhẹ của gọi thủ tục từ xa, thiết kế cho hiệu năng cao khi thực hiện các nội giao tiếp và các lời gọi từ các tiến trình khác nhau. Binder được thực hiện bằng cách tùy chỉnh lại Linux driver.
- **Services:** đã trình bày phía trên.
- **Intents:** là một đối tượng tin nhắn đơn giản, đại diện cho một ý định làm việc gì đó. Lấy ví dụ, khi một ứng dụng muốn hiển thị một trang web, nó gửi một “intent” cho hệ thống xử lý, hệ thống xác định một ứng dụng hoặc một đoạn mã nào đấy có khả năng xử lý.
- **ContentProvider:** một nơi chứa dữ liệu được truy cập trên thiết bị, một ứng dụng có thể cho phép các ứng dụng khác sử dụng dữ liệu của nó thông qua việc tạo ra các contentProvider.

1.3.3.4. Các API có thể phát sinh chi phí nhạy cảm

Một API có thể phát sinh chi phí là bất kỳ tính năng nào có thể tạo ra các khoản phí khi sử dụng nó. Hệ thống Android đưa các API này vào danh sách các API được hạn chế, người dùng phải chấp nhận cấp quyền cho các ứng dụng để sử dụng chúng. Bao gồm:

- Gọi điện thoại
- Gửi tin nhắn
- Dữ liệu di động
- Thanh toán trong ứng dụng
- Truy cập NFC

1.3.3.5. Thông tin thiết bị

Android cũng cố gắng hạn chế truy cập vào dữ liệu nhạy cảm, nhưng có thể gián tiếp cho biết các thuộc tính của người dùng, thói quen người dùng và cách sử dụng thiết bị. Mặc định các ứng dụng sẽ không được truy cập vào log của hệ thống, lịch sử trình duyệt, số điện thoại, thông tin định danh phần cứng, hay công mạng.

Nếu ứng dụng yêu cầu các thông tin trên, trình cài đặt sẽ hiện các thông báo, nếu người dùng không chấp nhận cấp quyền, ứng dụng sẽ không được cài đặt.

1.4. Mã độc trên Android

1.4.1. Hiện trạng

Cũng như trong thế giới của máy tính, các mã độc trên Android về bản chất cũng là các ứng dụng thông thường trên hệ thống, tuy nhiên được viết ra để thực hiện các hành vi độc hại gây thiệt hại cho người dùng, và hệ thống.

Các ứng dụng trên Android được cài chủ yếu từ các kho ứng dụng chính thống Chplay do Google quản lý, tất cả các ứng dụng khi được các nhà phát triển đưa lên đều phải qua kiểm soát của Google, tuy nhiên việc này cũng không thể đảm bảo hoàn toàn cho người dùng. Ngoài ra Android còn cho phép người dùng cài thêm các ứng dụng từ bên ngoài, các ứng dụng có thể tìm kiếm trên các công cụ tìm kiếm hầu hết đều có chứa mã độc. Đối với người dùng không có hiểu biết, việc cài một ứng dụng có cùng tên được tìm thấy đâu đó trên mạng internet được họ thực hiện một cách hoàn toàn tin tưởng thì việc thiết bị của họ bị lây nhiễm là rất cao.

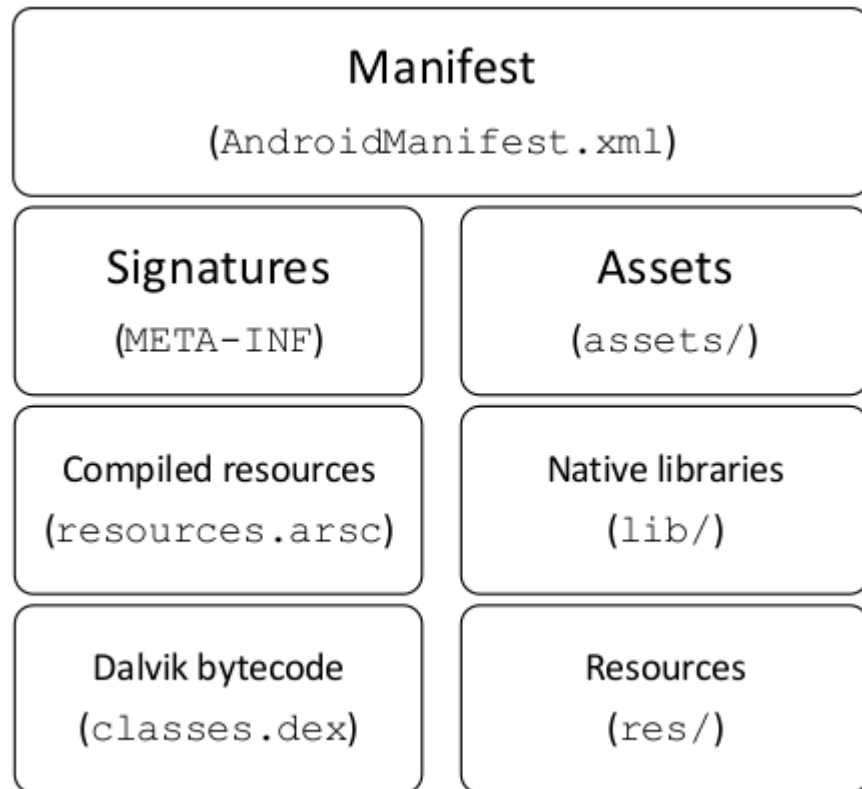
Các ứng dụng này chủ yếu thực hiện ba nhóm hành vi chính bao gồm: quảng cáo, ăn trộm thông tin người dùng, và thực hiện các tính năng phát sinh chi phí đối với người dùng. Đối với các ứng dụng quảng cáo, việc an gian với các nhà phát hành quảng cáo bằng cách liên tục hiển thị các quảng cáo và lừa cho người dùng bấm vào, làm ảnh hưởng nghiêm trọng đến trải nghiệm người dùng. Ngoài ra loại này còn thực hiện thêm việc cài thêm các ứng dụng độc khác lên máy người dùng. Đối với loại ăn cắp dữ liệu người dùng, hay còn được biết đến như Spyware, phần mềm gián điệp, loại mã độc này thường đọc tin nhắn, nhật ký cuộc gọi, sổ danh bạ, hoặc thậm chí là thực hiện ghi âm, chụp ảnh. Ảnh hưởng nghiêm trọng đến tính cá nhân của người dùng. Và cuối cùng, loại mã độc sử dụng các tính năng phát sinh cước phí gây ảnh hưởng đến tài chính của người dùng. Loại này thường tự gửi các tin nhắn đăng ký đến các dịch vụ, hoặc tính vi hơn là đăng ký hoặc thanh toán tiền cho các dịch vụ được các nhà mạng di động triển khai cơ chế thanh toán thông qua cơ chế truy cập vào các trang WAP của nhà mạng.

1.4.2. File thực thi trên Android

1.4.2.1. Cấu trúc

Như phần trên đã trình bày, file ứng dụng trên Android được biết đến là file với phần mở rộng .apk, phần này sẽ trình bày rõ hơn về file APK.

Hình 1.3 mô tả cấu trúc các thành phần chung của một tệp tin APK.



Hình 1.3 Cấu trúc tệp tin apk

- **Manifest:** chứa các thông tin về ứng dụng.
- **Signatures:** chứa các mã hash đã được ký bởi chữ ký của người phát triển, được sử dụng trong quá trình xác thực để đảm bảo tính toàn vẹn của ứng dụng.
- **Assets:** thư mục chứa các tài nguyên cần thiết như ảnh, tệp dữ liệu của ứng dụng, nơi có thể truy cập vào bằng AssetManager.
- **Resource.arsc:** Chứa các file đã được biên dịch trước như các file .xml.
- **Lib:** chứa các thư viện được sử dụng cho ứng dụng, bao gồm các file với phần mở rộng .so với native code.
- **Res:** nơi chứa các tài nguyên chưa được biên dịch.

- **Classes.dex:** file thực thi mã nguồn.

1.4.2.2. File DEX

Như đã trình bày, các ứng dụng chạy trên Android được viết chủ yếu bằng ngôn ngữ Java, quá trình từ mã nguồn Java đến khi mã được thực thi trên máy ảo Dalvik được mô tả trong hình 1.4 dưới đây.



Hình 1.4 Quá trình biên dịch

Mã nguồn java được biên dịch thành các file class. Thông thường đối với các ứng dụng java, sau khi được biên dịch thành bytecode, các file này sẽ được thực thi bởi máy ảo JVM, tuy nhiên để đảm bảo hiệu năng, và một số đặc thù khác, các file .class tiếp tục được biên dịch thành và gộp thành file .dex. File dex được thực thi bởi máy ảo Dalvik. Có một điểm quan trọng đối với file Dex. Android cho phép các ứng dụng có thể thực thi một file dex không có sẵn trong file cài đặt, và cũng không cần đặc quyền nào để thực hiện việc này. Điều này có nghĩa là, nếu ứng dụng mã hóa một phần mã nguồn ứng dụng, hoặc tải thêm phần mã nguồn này từ internet trong quá trình thực thi thì nó có thể thay đổi, bổ sung các tính năng hành vi vốn có của nó. Điều này là tốt, nhưng nếu nó được sử dụng cho mã độc thì việc kiểm soát các ứng dụng sẽ rất phức tạp. Trên thực tế đã có rất nhiều ứng dụng sử dụng cơ chế này để qua mặt Google khi đưa mã độc lên, điều này dẫn đến việc hàng trăm nghìn thiết bị bị nhiễm mã độc trước khi nó bị phát hiện và gỡ xuống. Phương pháp này kết hợp với java reflection trở thành một trong những trở ngại lớn nhất khi thực hiện phân tích tĩnh mã độc trên Android.

1.5. Dự án mã nguồn mở Android

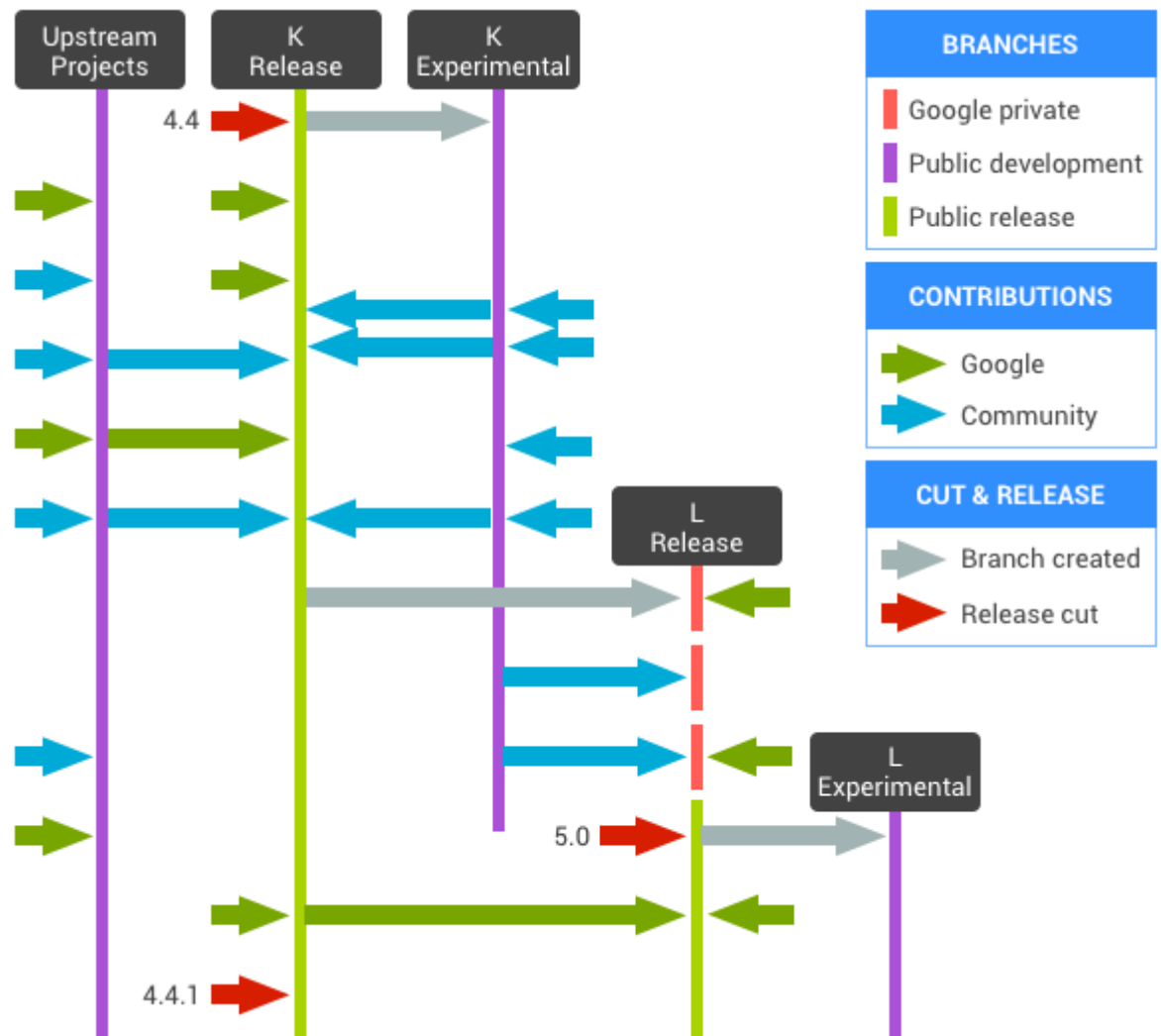
Dự án mã nguồn mở Android duy trì và phát triển một bộ phần mềm hoàn chỉnh và được chuyển tới các nhà phát triển phần cứng để họ triển khai trên các thiết bị của họ. Để đảm bảo chất lượng của Android, Google có các kỹ sư, các nhà quản

lý, thiết kế và kiểm tra chất lượng phần mềm và nhiều vị trí khác cần thiết để kiểm soát các sản phẩm này.

Do đó họ duy trì các nhánh phiên bản rõ ràng, giữa các phiên bản đã phát triển ổn định và các phiên bản đang trong quá trình phát triển.

1.5.1. Quản lý mã nguồn AOSP

Hình vẽ 1.5 dưới đây mô tả thiết kế phía sau trong việc quản lý mà phát hành các phiên bản của AOSP.



Hình 1.5 Các nhánh phát hành AOSP

- Tại bất kỳ thời điểm nào, đây luôn là bản phát hành mới nhất của nền tảng AOSP. Điều này thường biểu diễn dưới hình thức “cây”.
- Những nhà phát hành làm việc trên các nhánh này, sửa lỗi, chạy trên các thiết bị mới, nâng cao trải nghiệm người dùng và bổ sung các tính năng mới.

- Cùng với đó, Google làm việc trong nội bộ với các phiên bản tiếp theo của nền tảng Android. Họ phát triển các phiên bản mới bằng cách làm việc với các đối tác trên một thiết bị hàng đầu nào đó.
- Khi phiên bản tiếp theo được sẵn sàng phát hành, nó sẽ được đẩy công khai mã nguồn và trở thành bản mới nhất.

1.5.2. Tên mã, nhãn và số hiệu bản dựng

Android phát triển và phát hành các phiên bản mới sử dụng các tên được sắp xếp theo thứ tự bản chữ cái. Tên mã ứng với các số hiệu bản dựng, cùng với các mức API và NDK được thể hiện chi tiết ở bảng 1.3.

Bảng 1.3 Mức API theo phiên bản AOSP

Code name	Version	API level
Oreo	8.1.0	API level 27
Oreo	8.0.0	API level 26
Nougat	7.1	API level 25
Nougat	7.0	API level 24
Marshmallow	6.0	API level 23
Lollipop	5.1	API level 22
Lollipop	5.0	API level 21
KitKat	4.4 - 4.4.4	API level 19
Jelly Bean	4.3.x	API level 18
Jelly Bean	4.2.x	API level 17
Jelly Bean	4.1.x	API level 16
Ice Cream Sandwich	4.0.3 - 4.0.4	API level 15, NDK 8
Ice Cream Sandwich	4.0.1 - 4.0.2	API level 14, NDK 7
Honeycomb	3.2.x	API level 13
Honeycomb	3.1	API level 12, NDK 6
Honeycomb	3.0	API level 11
Gingerbread	2.3.3 - 2.3.7	API level 10
Gingerbread	2.3 - 2.3.2	API level 9, NDK 5
Froyo	2.2.x	API level 8, NDK 4

Eclair	2.1	API level 7, NDK 3
Eclair	2.0.1	API level 6
Eclair	2.0	API level 5
Donut	1.6	API level 4, NDK 2
Cupcake	1.5	API level 3, NDK 1
(no code name)	1.1	API level 2
(no code name)	1.0	API level 1

Bắt đầu từ Donut, danh sách chính xác các nhãn và bản dựng được thể hiện trong bảng bảng 1.4 (một số nhánh):

Bảng 1.4 Nhãn các nhánh AOSP

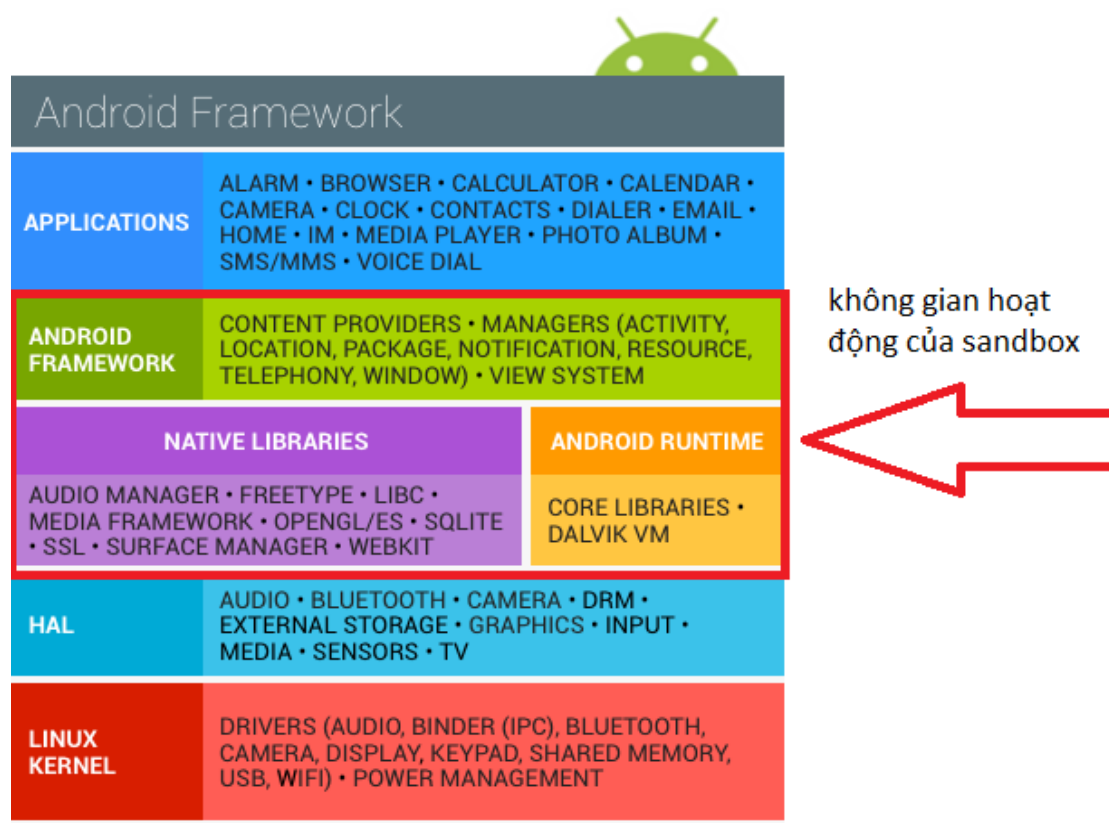
Build	Branch	Version	Supported devices
OPM5.171019.017	android-8.1.0_r18	Oreo	Nexus 5X, Nexus 6P
OPM3.171019.016	android-8.1.0_r17	Oreo	Nexus 5X, Nexus 6P
OPM1.171019.022.A1	android-8.1.0_r16	Oreo	Pixel C
OPM1.171019.021	android-8.1.0_r15	Oreo	Pixel 2 XL, Pixel 2, Pixel XL, Pixel
OPM5.171019.015	android-8.1.0_r14	Oreo	Nexus 5X, Nexus 6P
OPM3.171019.014	android-8.1.0_r13	Oreo	Nexus 5X, Nexus 6P

CHƯƠNG 2. THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG GIÁM SÁT HÀNH VI MÃ ĐỘC TRÊN HỆ ĐIỀU HÀNH ANDROID

2.1. Thiết kế hệ thống

2.1.1. Nguyên lý làm việc

Tổng thể hệ thống được thiết kế nhằm mục đích giám sát các API của hệ thống khi được một ứng dụng nào đó gọi đến, trong trường hợp này các mã độc. Để làm được việc này, ta sử dụng cách chỉnh sửa lại mã nguồn hệ thống API trong AOSP, khi đó từng API được giám sát, ngoài việc thực hiện việc như nó đã được thiết kế trước đó, nó sẽ làm thêm nhiệm vụ đẩy một thông điệp qua logcat.

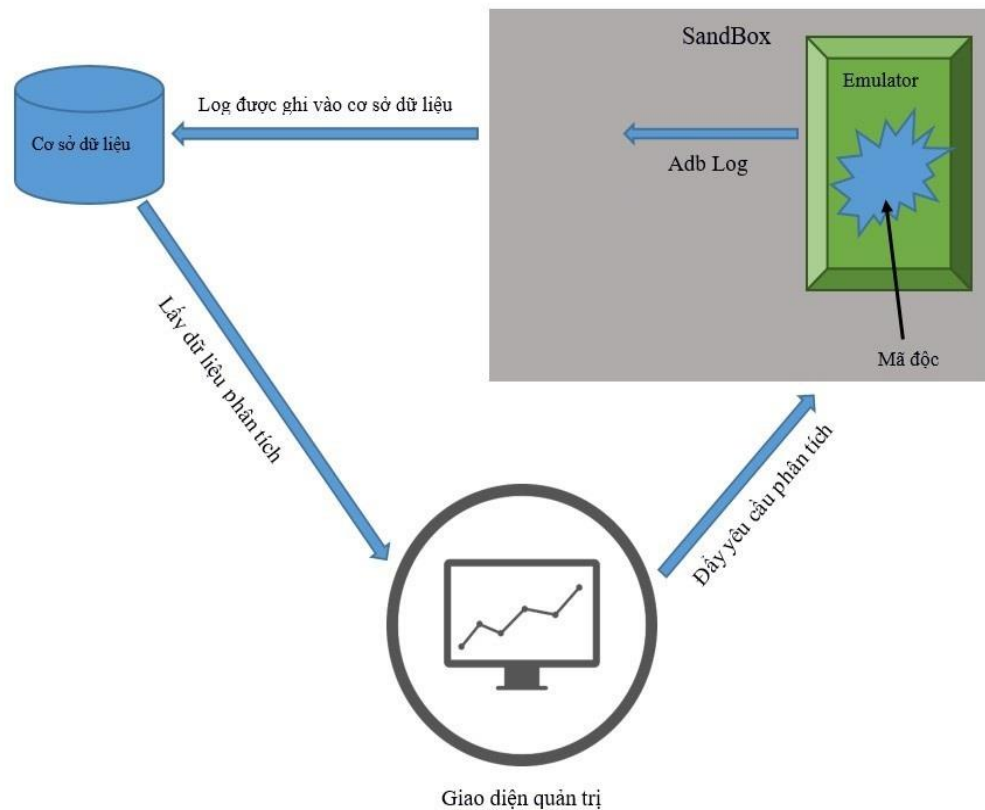


Hình 2.1 Không gian hoạt động của Sandbox

Trong mô hình hệ thống Android, các phần mã nguồn được chỉnh sửa nằm trong thành phần android framework và các thư viện chuẩn của android. Khi mã độc chạy ở tầng ứng dụng cao nhất trong hệ thống, các lời gọi sẽ được chuyển xuống cho tầng ngay bên dưới. Khi này các lời gọi sẽ được “bắt lấy”, đảm bảo các hoạt động của mã độc được giám sát hoàn toàn. Ngoài ra, hoạt động theo cơ chế này, sẽ đảm bảo được tính trong suốt với mã độc. Khi phân tích các mã độc, việc này không

làm lộ ra rằng nó đang chạy trong một hệ thống đang được theo dõi bằng phương pháp trên.

2.1.2. Mô hình hệ thống



Hình 2.2 Mô hình hệ thống

Hệ thống gồm ba thành phần chính

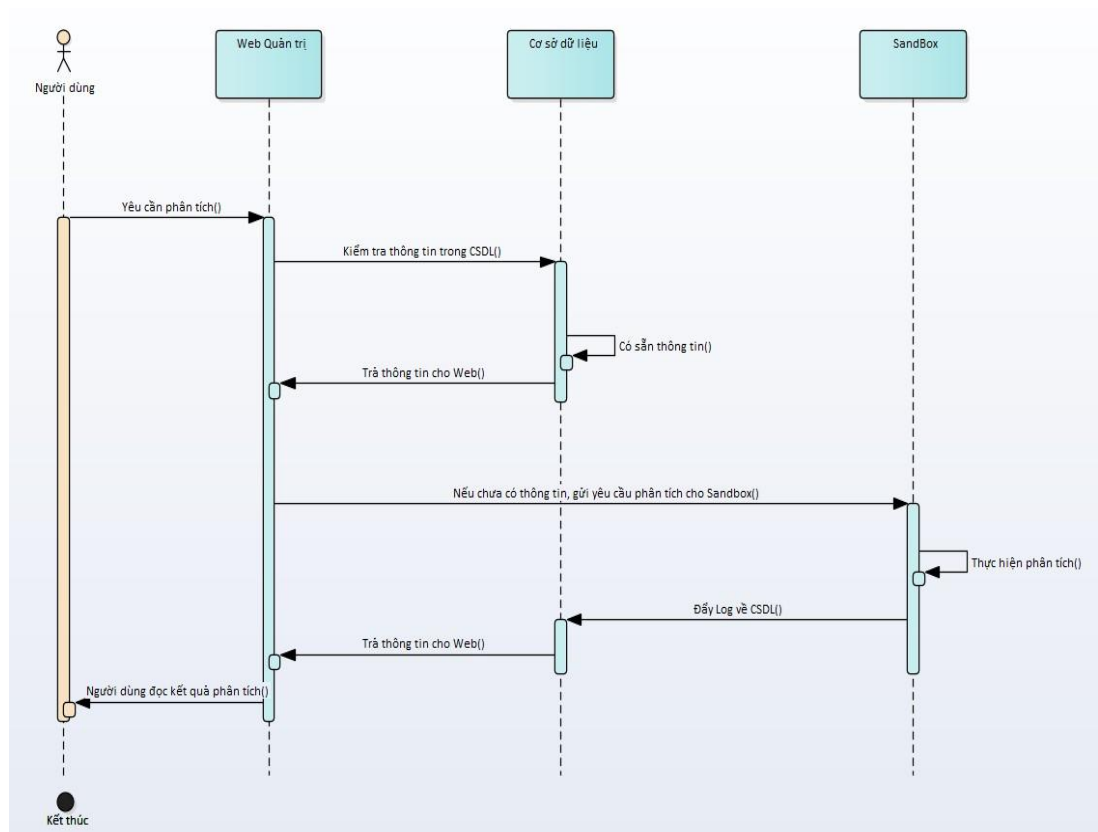
- Sandbox: thực thi mã độc, phân tích log, đẩy log về cơ sở dữ liệu
- Cơ sở dữ liệu: lưu trữ dữ liệu
- Website quản trị: tương tác với người dùng, nhận yêu cầu và trả kết quả phân tích

2.1.3. Tương tác giữa các thành phần trong hệ thống

Để bắt đầu hoạt động, người sử dụng thông qua giao diện website quản trị để đưa mẫu vào hệ thống. Hệ thống kiểm tra một số thông tin cơ bản của mẫu như giá trị hàm băm, để kiểm tra xem mẫu đã được phân tích trước đó hay chưa. Tiếp theo, nếu mẫu chưa có trong hệ thống thì sẽ được đem đi phân tích.

Quá trình phân tích diễn ra như sau:

1. Mẫu được phân tích tĩnh để lấy một số thông tin cơ bản bao gồm: giá trị hàm băm, các thông tin trong tệp `manifest`, các tài nguyên đi kèm trong mẫu
2. Mẫu được cài đặt vào máy ảo android, và khởi chạy thông qua adb
3. Log được đẩy về khối nhận log trên hệ thống rồi đẩy về cơ sở dữ liệu
4. Sau khi log được đẩy về hệ thống, từ giao diện web người phân tích có được kết quả phân tích



Hình 2.3 Tương tác với người dùng

2.1.4. Chi tiết hoạt động của hệ thống phân tích

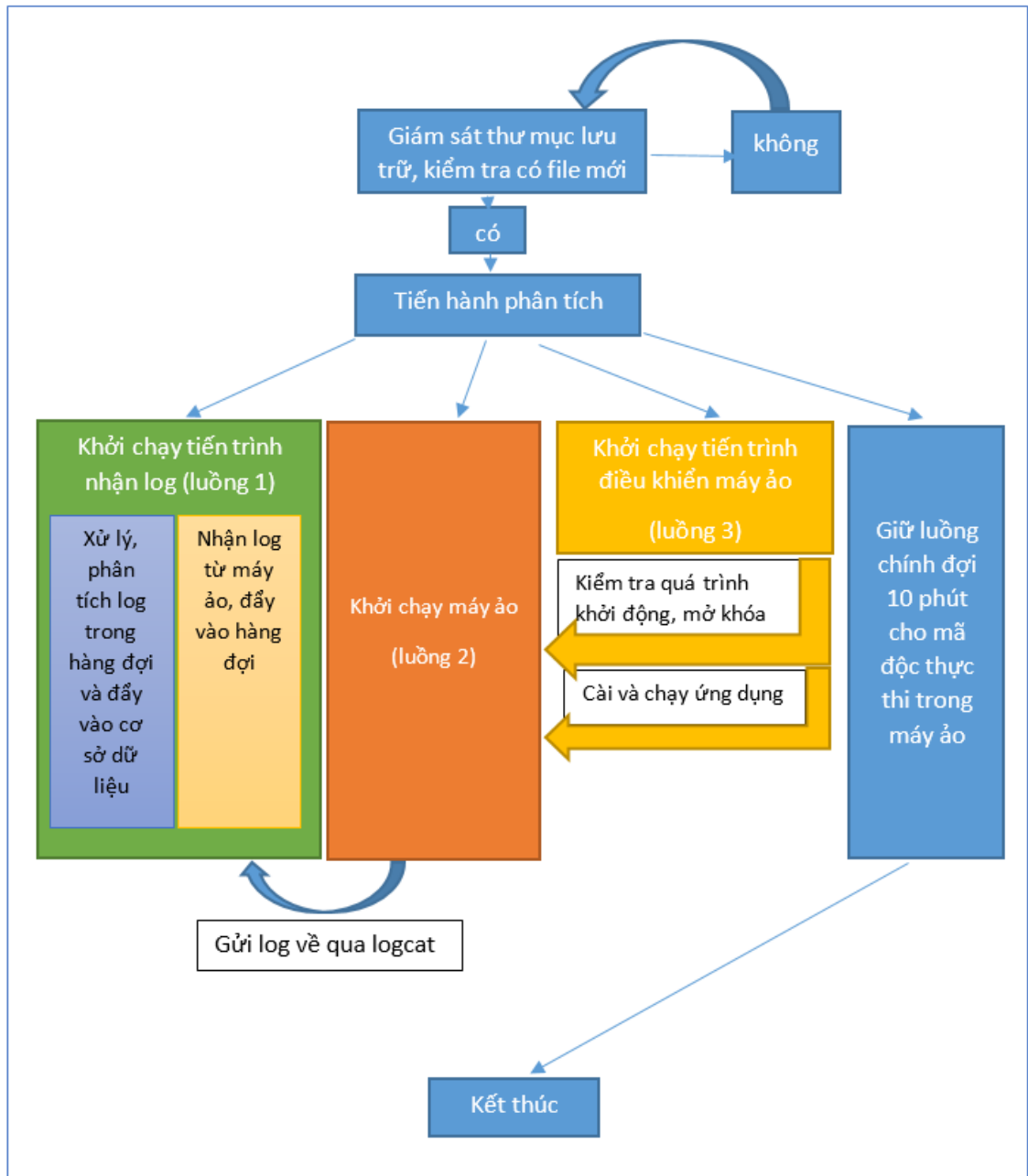
Sẽ có một thư mục chứa các tệp cần phân tích, hệ thống sẽ giám sát thư mục này, các file trong thư mục lần lượt được đem đi phân tích. Trước khi một tệp được đem đi phân tích nó sẽ được xác định xem đã có một phiên bản trước đó của nó đã được phân tích chưa bằng cách tính mã hash(ở đây sử dụng sha256) để kiểm tra xem

có thông tin về mẫu này trong cơ sở dữ liệu chưa. Nếu chưa có thông tin, mẫu sẽ được phân tích.

Trước khi đưa vào máy ảo, mẫu sẽ được trích xuất các thông tin cơ bản thông qua việc phân tích tĩnh. Các thông tin bao gồm: mã sha256, mã md5, tên, tên gói (packageName), các quyền, các service . . . Sau đó hệ thống khởi chạy thêm 3 luồng thực thi bao gồm: Máy ảo, điều khiển máy ảo, xử lý log. Trước tiên, máy ảo được khởi động, luồng điều khiển máy ảo điều khiển máy ảo qua adb, nó sẽ kiểm tra xem khi nào hệ thống khởi động xong thông qua event `BOOT_COMPLETED`. Sau khi khởi động xong, luồng điều khiển sẽ gửi các input giả lập để mở khóa màn hình, tiếp theo đó là cài ứng dụng lên máy ảo và khởi chạy ứng dụng đó. Sau khi quá trình này hoàn thành luồng điều khiển sẽ tự kết thúc. Luồng nhận và xử lý log bao gồm 2 luồng hoạt động khác nhau.

- Luồng thứ nhất có nhiệm vụ nhận log từ máy ảo thông qua adb, và đẩy vào hàng đợi.
- Luồng thứ 2 thực hiện phân tích, lọc các thông tin cần thiết và đẩy vào cơ sở dữ liệu.

Lý do cần phải tách thành 2 luồng khác nhau, mà không thực hiện trong cùng một luồng là vì do việc phân tích log, lọc log, và đặc biệt là đẩy log vào cơ sở dữ liệu có thể mất nhiều thời gian, trong khi đó log được đẩy về từ máy ảo là rất nhiều, và liên tục. Do vậy việc sử dụng hàng đợi với 2 luồng khác nhau sẽ giảm thiểu khả năng mất log. Chi tiết các luồng thực thi được mô tả ở hình 9 với các luồng thực thi khác nhau



Hình 2.4 Luồng hoạt động của hệ thống

2.1.5. Các công nghệ sử dụng

2.1.5.1. Sandbox

Bao gồm ba thành phần nhỏ.

- Máy ảo
- Adb
- Trình xử lý log

Sử dụng máy ảo có sẵn đi theo trong bộ SDK do google cung cấp, máy ảo này có nhiệm vụ giả lập phần cứng, khởi chạy hệ điều hành. Adb là công cụ cầu

nổi trong quá trình gỡ lỗi, nhận log từ logcat của Android, adb cũng dùng công cụ có sẵn được kèm theo trong SDK. Thành phần cuối cùng là một công cụ được viết bằng python có nhiệm vụ nhận log từ logcat thông qua adb để đẩy vào CSDL.

2.1.5.2. Cơ sở dữ liệu

Nơi lưu trữ tập trung thông tin các mẫu sau khi được phân tích, đồng thời trả dữ liệu khi được yêu cầu từ web quản trị. Ở đây sử dụng nền tảng noSQL với **MongoDB**.

2.1.5.3. Web quản trị

Nơi cung cấp giao diện người dùng, tiếp nhận các yêu cầu phân tích và chuyển qua sandbox thực hiện. Đồng thời thể hiện kết quả phân tích cho người dùng.

- Webserver được xây dựng trên nền tảng **Flask** là một FrameWork được dùng để xây dựng máy chủ web (ngôn ngữ python).
- WebClient sử dụng công nghệ **AngularJS** xây dựng ứng dụng web đơn trang, hiển thị và tạo các bộ lọc, hiển thị log với giao diện cho người dùng.

2.2. Xây dựng hệ thống

2.2.1. Xây dựng sandbox phân tích

2.2.1.1. Biên dịch thử nghiệm mã nguồn chuẩn

Để tiến hành biên dịch rom, việc đầu tiên cần thực hiện là tải về bộ mã nguồn Android. Dự án mã nguồn mở Androi được google cung cấp trong một kho chứa (repositories) tại <https://android.googlesource.com/>

a) Yêu cầu

Việc build rom từ AOSP cần đáp ứng một số yêu cầu đối với các thiết bị phân cứng và phần mềm sau:

- *Phần cứng:*
 - Môi trường 64bit đối với các phiên bản 2.3.x và các phiên bản mới hơn
 - Tối thiểu 100GB lưu trữ mã nguồn và mở rộng lên 150GB khi thực hiện

biên dịch, trong trường hợp biên dịch cho nhiều thiết bị và có sử dụng ccache thì cần thêm bộ nhớ lưu trữ.

- Nếu đang chạy linux trong một máy ảo, phải đảm bảo tối thiểu 16GB RAM/swap.

- *Phần mềm*

- Nhánh master của AOSP trên git

- *Hệ điều hành và JDK*

- Nếu đang phát triển AOSP trên nhánh master thì cần phải sử dụng các phiên bản hệ điều hành : Ubuntu 14.04 hoặc Mac OS v10.10 hoặc mới hơn.
- Java Development Kit.

- *Các gói phần mềm*

- Python 2.6 -- 2.7 từ python.org
- GNU Make 3.81 -- 3.82 từ gnu.org
- Git 1.7 hoặc mới hơn từ git-scm.com

b) Chuẩn bị môi trường

- *Chọn nhánh*

Đối với các phiên bản cần biên dịch sẽ có các nhánh khác nhau, cần phải chỉ rõ. Danh sách các nhánh, tên nhân cho các phiên bản được liệt kê tại <https://source.android.com/setup/build-numbers>

- *Cài đặt môi trường biên dịch trên linux*

Các hướng dẫn dưới đây áp dụng cho tất cả các nhánh, bao gồm cả nhánh master

Cài đặt JDK

```
$sudo apt-get update
$sudo apt-get install openjdk-7-jdk
```

- *Cài đặt các gói phần mềm yêu cầu*

Để quá trình biên dịch hoàn tất, một số gói phần mềm bổ sung cần được cài đặt

```
$sudo apt-get install git-core gnupg flex bison gperf build-essential zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z-dev ccache libgl1-mesa-dev libxml2-utils xsltproc unzip
```

- *Sử dụng các thư mục lưu trữ cho các lần biên dịch khác nhau*

Mặc định, các tệp sau khi build sẽ được lưu trong thư mục out/ trong thư mục gốc của AOSP. Trong một số máy với nhiều thiết bị lưu trữ, việc build sẽ nhanh hơn khi lưu các file nguồn và các tệp sai khi build ở các phần khác nhau. Để tăng hiệu năng, thư mục out/ nên được đặt trong thiết bị lưu trữ đã được tối ưu hóa tốc độ, tất cả các tệp tin đều được tạo lại trong trường hợp hệ thống tệp tin bị lỗi.

Để cài đặt việc này, cần xác định biến `OUT_DIR_COMMON_BASE`, biến sẽ lưu thư mục đầu ra.

```
$export OUT_DIR_COMMON_BASE=<path-to-your-out-directory>
```

Thư mục lưu trữ cho các phần riêng biệt của mã nguồn sẽ được đặt tên theo thư mục mã nguồn đó.

Ví dụ `/source/master1/` sẽ được lưu lại `/output/master1`

- *Tối ưu môi trường (tùy chọn)*

Việc tối ưu lại môi trường biên dịch bằng cách sử dụng ccache với các trình biên dịch cho C/C++ có thể tăng tốc độ biên dịch cho những lần thực hiện sau. Việc này rất hữu ích trong trường hợp cần chạy lệnh `$make clean` hoặc `$make clobber` hay việc phải chuyển qua lại giữa các nhánh khác nhau.

Để sử dụng ccache, thực hiện các lệnh như sau

```
$export USE_CCACHE=1
$export CCACHE_DIR=/<path_of_your_choice>/ccache
$prebuilts/misc/linux-x86/ccache/ccache -M 50G
```

Việc build AOSP được đề nghị với 50GB cho thư mục cache.

Thêm vào nội dung sau vào tệp `.bashrc` hoặc tương đương

```
$export USE_CCACHE=1
```

c) Tải mã nguồn

Android được lưu trữ trong kho chứ được cung cấp bởi google(Git repository). Git repository này bao gồm cả các metadata cho mã nguồn Android như các thay đổi trong mã nguồn, ngày phát hành.

- *Cài đặt repo*

Để cài đặt repo, cần đảm bảo đã có thư mục `/home/bin`

```
$mkdir ~/bin  
$PATH=~/bin:$PATH
```

- *Tải về repo tool*

```
$curl https://storage.googleapis.com/git-repo-downloads/repo >  
~/bin/repo  
$chmod a+x ~/bin/repo
```

- *Khởi tạo máy khách repo*

Sau khi tải repo tool, cài đặt các thông tin cho trình khách để có thể truy cập tài nguyên.

Tạo thư mục trống, nơi sẽ lưu trữ mã nguồn

```
$mkdir WORKING_DIRECTORY  
$cd WORKING_DIRECTORY
```

Cài đặt thông tin người dùng cho Git

```
$git config --global user.name "Your Name"  
$git config --global user.email "you@example.com"
```

Init repo

```
$repo init -u https://android.googlesource.com/platform/manifest
```

Lựa chọn nhánh mã nguồn. Ví dụ lệnh sau sẽ chọn phiên bản Android 4.0.1 release 1

```
$repo init -u https://android.googlesource.com/platform/manifest  
-b android-4.0.1_r1
```

- *Tải mã nguồn android*

Để bắt đầu quá trình tải thực hiện lệnh sau

```
$repo sync
```

Việc này cần một khoảng thời gian để tải xong.

d) Tiến hành chạy biên dịch

- *Xóa lần build trước.*

```
$make clobber
```

- *Khởi tạo môi trường.*

Khởi tạo môi trường với mã trong `envsetup.sh`

```
$source build/envsetup.sh
```

Hoặc

```
$build/envsetup.sh
```

- *Chọn thiết bị đầu cuối.*

```
$lunch aosp_arm-eng
```

Trong trường hợp chạy lệnh `lunch` mà không có tham số đầu vào, một danh sách các thiết bị cuối sẽ được hiển thị cho phép lựa chọn trực tiếp.

- *Bắt đầu biên dịch*

Biên dịch tất cả với make. GNU có thể xử lý song song nhiều luồng cùng một lúc với tham số `-jN` trong đó N là số luồng thường gấp 1, 2 lần số lõi vật lý của vi xử lý đang chạy của hệ thống. Ví dụ đối với các CPU 4 lõi, 2 luồng trên 1 lõi thì để tối ưu nên chạy make với tham số `-j4` hoặc `-j8`

```
$make -j4
```

Việc biên dịch toàn bộ mã nguồn AOSP cần nhiều thời gian, trong lần chạy

thử nghiệm với hệ thống bao gồm Ubuntu 16.04.3 LTS, Intel Core I3-8100(4 cores, 4 thread) 8GB RAM với make -j4 cần 4 giờ để hoàn thành.

```
#### make completed successfully (03:50:51(hh:mm:ss)) ####
```

2.2.1.2. Chỉnh sửa bộ thư viện API trong hệ thống Android

a) Xác định các API cần sửa

Trước hết việc chỉnh sửa mã nguồn trong *android framework* và thư viện *java SDK* là nhằm mục đích thu thập các thông tin, hành vi của mã độc trên hệ thống. Vì vậy việc đầu tiên là cần xác định các API nào cần chỉnh sửa. Để làm việc này cần hiểu rõ chức năng của các mã độc đã biết, các hình vi thường gặp ở các mã độc. Hiện tại chia làm một số nhóm API như sau:

- Nhóm API về kết nối mạng
- Nhóm API về tin nhắn SMS
- Nhóm API về thao tác với file
- Nhóm API về lấy thông tin hệ thống
- Nhóm API về thao tác với cơ sở dữ liệu
- Nhóm API về vị trí người dùng
- Nhóm API về theo dõi môi trường người dùng (ghi âm, ghi hình)
- Nhóm API về chạy mã bổ sung(load Dex) và chạy shell code(cmd)

Phân vùng mã nguồn sẽ được rà soát kiểm tra và chỉnh sửa trong bộ mã nguồn AOSP nằm trong các đường dẫn sau:

```
├── frameworks
│   ├── base
│   └── opt
├── libcore
│   ├── luni
│   └── src
│       ├── main
│       └── java
```

Nhóm các API cho framework Android nằm trong đường dẫn

Framework/base và Framework/opt

Nhóm các API chuẩn của Java nằm tại đường dẫn

lib/luni/src/main/java

b) Sửa API với mục đích ghi log

Để bắt đầu chỉnh sửa, cần phải hiểu để thực hiện một chức năng thì nó phải gọi API gì của hệ thống. Ví dụ để thực hiện gửi một tin nhắn văn bản tới một số nào đó ta có đoạn mã sau:

```
try{
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage("123456789",null,"Sent from
    Android",null,null);
}
catch(Exception e){
    Log.d("test", "Cannot send message");
}
```

Ta thấy khi muốn gửi một tin nhắn đi, bắt buộc phải gọi phương thức `sendTextMessage()` do vậy để thử nghiệm ta sẽ sửa lại mã nguồn của phương thức `sendTextMessage()`

```
public void sendTextMessage(
    String destinationAddress, String scAddress,
    String text,
    PendingIntent sentIntent, PendingIntent
    deliveryIntent){
    System.out.println("type: 'sms',
    apiName: 'android.telephony.SmsManager.sendTextMessage',params: { '
    " + destinationAddress + "','"+ scAddress + "','"+ text + "'" );
    sendTextMessageInternal(destinationAddress, scAddress,
    text,
        sentIntent, deliveryIntent, true /*
    persistMessageForCarrierApp*/);
}
```

Sau khi sửa lại mã nguồn như trên, ta tiến hành biên dịch ROM

`$make -j4`

Quá trình build hoàn tất, khởi động máy ảo với ROM đã build ở bên trên

`$emulator`

Máy ảo đã khởi động, ta vào phần chức năng tin nhắn mặc định của máy, gửi

đi một tin nhắn, kết quả thu được ở logcat ta có như sau

```
03-01 10:39:57.419 1158 1451 I System.out: type:'sms',
apiName:'android.telephony.SmsManager.sendMultipartTextMessage',
params:{'333','null','[This is a message ]'}
03-01 10:39:57.419 1158 1451 I System.out: type:'sms',
apiName:'android.telephony.SmsManager.sendTextMessage',params:{'
333','null','This is a message '}
```

Như vậy ta thử nghiệm thành công việc ghi log cho một API cùng với các tham số truyền vào.

Để tiếp tục, cần hoàn thành sửa toàn bộ các API ta cần theo dõi với các bước như trên cho các API khác.

Ở đây có một lưu ý, đối với các API chỉnh sửa mà thay đổi, thêm, hay bớt các phương thức trong một class nào đó, trước khi thực hiện biên dịch lại ta cần phải update lại các API bằng cách

```
$make update-api
```

Sau đó mới chạy

```
$make -j4
```

c) Định dạng log

Để thuận tiện trong việc lấy thông tin và phân tích sau này, các API cần phải thống nhất chung một định dạng để hệ thống phân tích sau này có thể truy xuất, phân tích dễ dàng.

Ở đây phương pháp lựa chọn là chuẩn hóa theo định dạng dữ liệu json như sau:

```
Object{
  Hash: "value",
  Type: "network | file | service | sms ...",
  ApiName: "apiname",
  Params: {param1: "value", ...},
  PID: "id",
  TID: "tid"
}
```

Với:

Type: là nhóm các API, bao gồm nhóm các được liệt kê ở mục 2.2.1.2 a).

Hash: giá trị địa hash của tệp đang phân tích hiện tại.

ApiName: định danh cho api được gọi, đặt theo các lớp và phương thức.

params: là các tham số truyền vào khi API được gọi.

PID: id của tiến trình sinh ra log.

TID: id của luồng sinh ra log.

return: giá trị của hàm trả về nếu cần.

Danh sách các API được giám sát được liệt kê ở phần phụ lục

d) Lưu ý khi lấy giá trị của các tham số

Trong quá trình thực hiện viết mã cho một API nào đây cần theo dõi, ta cần lấy giá trị của các tham số truyền vào. Khi này có thể xảy ra trường hợp các tham số có giá trị không hợp lệ, nếu không xử lý đúng có thể gây lỗi ứng dụng hoặc tệ hơn, các API được hệ thống sử dụng bị lỗi có thể gây lỗi hệ điều hành kết quả là hệ thống bị treo hoặc tệ hơn là không thể khởi động được. Để kiểm soát việc này ta cần tính cẩn thận tất cả các khả năng có thể xảy ra với từng API cụ thể. Để ví dụ cụ thể ta xem xét một API sau: `URLConnection(URL url)`

Khi một ứng dụng gọi đến lớp **URLConnection**, hàm khởi tạo được truyền vào một URL chứa địa chỉ cần kết nối đến, để ghi log thông thường sẽ như sau

```
URLConnection(URL url) {  
    Log_function(url.toString());  
}
```

Khi thực hiện như trên có thể xảy ra ngoại lệ khi url truyền nào là null. Do vậy để đảm bảo an toàn, ta cần phải kiểm tra kỹ giá trị của tham số, hoặc dùng một số hàm an toàn có thể như `String.valueOf()`. Đây là vấn đề cần phải lưu ý khi thực hiện, bởi lẽ nếu xảy ra trường hợp trên làm hệ thống không khởi động được thì sẽ rất khó để tìm ra đâu là đoạn mã gây lỗi.

trình build, thay vì để các tham số mặc định và bị mã độc phát hiện, ta cần sửa lại thành thông tin của một thiết bị thật, việc này được thực hiện bằng cách mở file.

```
Build/buildtool.sh
```

Và sửa các tham số như sau:

```
echo "ro.hardware=hammerhead"
echo "ro.bootloader=HHZ11k"
echo "ro.product.model=Nexus 5"
echo "ro.product.brand=google"
echo "ro.product.name=hammerhead"
echo "ro.product.device=hammerhead"
echo "ro.product.board=hammerhead"
```

Sau khi sửa đổi như trên, máy ảo sẽ có thông các tham số như thiết bị vật lý thật, trong trường hợp mã độc sử dụng các hàm để lấy thông tin như `Build.BROAD` hay `Build.DEVICE` sử dụng mã java thuần hoặc gọi trực tiếp từ mã C/C++ `system.getProperty(Build.DEVICE)` sẽ bị đánh lừa bởi thông tin trên và tiếp tục thực thi.

2.2.2. Xây dựng thành phần điều khiển máy ảo Android

Sau khi có được máy ảo Android với bản ROM được chỉnh sửa theo ý muốn. Ta cần xây dựng một thành phần có chức năng quản lý, thực hiện các công việc bao gồm: khởi động, tắt, giả lập các đầu vào, cài ứng dụng, khởi động ứng dụng. Để làm được các tính năng trên, một đoạn mã Python kết hợp sử dụng adb để làm cầu nối điều khiển.

2.2.2.1. Khởi chạy máy ảo với Emulator

Để chạy được máy ảo, ta cần sử dụng đến công cụ giả lập máy ảo Android Emulator được đi kèm với AOSP được lưu tại đường dẫn:

```
~AOSP/prebuilts/emulator/emulator
```

Từ Python ta gọi đến Emulator thông qua mã sau:

```
emulator -sysdir ~/out
```

Trong đó thư mục `out` là thư mục chứa bản ROM đã build được ở trên. Với python ta sử dụng subprocess và thiết lập stdout của luồng thành PIPE để có thể chạy một luồng mới

2.2.2.2. Kiểm soát quá trình khởi động, giả lập đầu vào và cài ứng dụng

Trong hệ thống Android, một khi hệ thống khởi động sau nó sẽ gửi đi một thông điệp thông qua broadcast với tham số `BOOT_COMPLETED`. Để xác định được khi nào hệ thống khởi động xong ta cần bắt lấy thông điệp này. Bằng công cụ adb ta mở thêm một luồng mới với tham số `-logcat` và kiểm tra với cụm từ `BOOT_COMPLETED`, sau khi bắt được sự kiện này ta sẽ đóng luồng logcat hiện tại và mở thêm một luồng mới để cài ứng dụng, đồng thời kiểm tra xem việc cài ứng dụng có thành công hay không.

```
Adb install path/to/apk
```

Path/to/apk là đường dẫn đến file ta cần phân tích. Output của lệnh trên sẽ cho ta biết kết quả việc có cài thành công hay không. Sau khi cài thành công, việc tiếp theo cần làm đó là khởi chạy ứng dụng. Để khởi động ứng dụng ta tiếp tục sử dụng adb như sau:

```
# shell code to start application
os.system(config.adb.adbPath + " shell monkey -p " +
self.packageName
+ " -c android.intent.category.LAUNCHER 1")
```

Lệnh này có tác dụng mở activity mặc định của ứng dụng. Sau khi ứng dụng được cài thành công, luồng này sẽ được ngắt.

2.2.2.3. Nhận và phân tích log

Để thực hiện việc nhận, phân tích log ta vẫn sử dụng công cụ adb với tham số `logcat` để lấy được log được đẩy về từ emulator thông qua logcat. Như đã phân tích, việc này cần được xử lý trong hai luồng khác nhau và một hàng đợi với cơ chế FIFO, cơ chế hàng đợi này được python hỗ trợ với Queue. Định dạng log được gửi về đã được ta định trước, ngoài ra còn lẫn cả log của hệ thống và của ứng dụng khác, để lọc được log cần thiết đã được thiết kế. Một biểu thức chính quy được sử dụng để kiểm tra xem liệu một dòng có phải là dòng log mà ta cần không như sau

```
pattern = re.compile("(^[0-9-]+ [0-9:~.]+) [ ]+([0-9]+) [ ]+([0-9]+) ([A-Z]) (.*)$")
```

Trong đó các số đầu tiên xác định cho ngày giờ, định danh các tiến trình, các luồng hoạt động nơi sinh ra log, a-z xác định cho mức độ, loại log như info, debug, warning ...

Tiếp theo đó là xác định liệu đây có phải là log của hệ thống API được giám sát hay không, lưu ý rằng do sử dụng `system.out.print` nên hệ thống log sẽ luôn có dạng bắt đầu bằng “systemout: ‘type’: ...” sau khi tách được thông tin như đã định trước, việc còn lại là đổi log trên về dạng dữ liệu json và đẩy vào cơ sở dữ liệu.

```
try:
    j = json.loads('{ ' + g + ' }')
    self.db.insert_log({'hash': self.hash_value, 'time':
datetime.now(), 'pid': int(pid), 'tid': int(tid),
                        'type': j['type'], 'apiName':
j['apiName'], 'params': j['params']})
except ValueError as E:
    pass
```

Đoạn mã tạo các luồng thực thi:

```
# start emulator
emulator_thread = Thread(target=self.start_emulator)
emulator_thread.daemon = True
emulator_thread.start()

# start parser
parser_thread = Thread(target=self.handler_queue)
parser_thread.daemon = True
parser_thread.start()

# start receive log
receive_thread = Thread(target=self.start_receive)
receive_thread.daemon = True
receive_thread.start()

# waiting for emulator start

while not self.boot_completed or not self.install_apk_done:
    sleep(1)

# sleep
print "waiting for analysis 10*60 seconds"
sleep(5 * 60)
self.kill = True
sleep(5)
```


Nếu quá trình khởi động các luồng thành công, hệ thống sẽ đợi cho máy ảo chạy 5 phút rồi tắt, là khoảng thời gian cho mã độc thực hiện các hành vi của nó.

2.2.2.4. Xây dựng hệ thống lưu trữ log

Hệ thống cơ sở dữ liệu được lựa chọn sử dụng ở đây là hệ quản trị cơ sở dữ liệu MongoDB trên nền tảng noSQL. Một trong những lý do lựa chọn như vậy là do việc lưu trữ log được sinh ra ở các API là khác nhau, các bản ghi được lưu trữ ở dạng object với các trường dữ liệu ở nhiều kiểu khác nhau. Thêm vào nữa là hệ thống hiện tại đang trong quá trình nghiên cứu và phát triển thêm, các trường dữ liệu khi được lưu trữ vào cơ sở dữ liệu ở hiện tại và tương lai sau này có thể có khác biệt khi hệ thống phát triển, việc thay đổi cấu trúc dữ liệu sẽ kèm theo nhiều việc không cần thiết. MongoDB là nền tảng miễn phí được sử dụng rộng rãi

2.2.2.5. Cài đặt MongoDB

Trên nền tảng Linux cụ thể là Ubuntu 16.04 LTS việc cài đặt MongoDB thực hiện như sau

- Lấy khóa công khai được sử dụng cho hệ thống quản lý các gói

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5
```

- Khởi tạo danh sách file cho mongodb

```
echo "deb [ arch=amd64,arm64 ]  
https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.6  
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-  
3.6.list
```

- Tải lại các gói phần mềm cục bộ

```
sudo apt-get update
```

- Cài đặt gói phần mềm MongoDB

```
sudo apt-get install -y mongodb-org
```

Sau khi việc cài đặt hoàn tất, khởi động MongoDB bằng câu lệnh

```
sudo service mongod start
```

2.2.2.6. Đánh index cho các trường dữ liệu

Nhằm tăng tốc việc truy xuất dữ liệu sau này cho các tác vụ như tìm kiếm, lấy thông tin. Việc đánh index giúp hệ thống dữ liệu quản lý và truy xuất dữ liệu nhanh hơn sau này. Hãy quan sát một document sau:

```
{
  "_id": ObjectId("5acdc5b84f76484f122163e3"),
  "hash": "testhash",
  "pid": 24031,
  "params": {
    "key": "[90, 117, 77, 102, 87, 67, 105, 55]"
  },
  "apiName": "javax.crypto.spec.DESKeySpec.DESKeySpec",
  "time": ISODate("2018-04-11T15:22:16.771Z"),
  "tid": 24031,
  "type": "crypto"
}
```

Các trường: hash, pid, tid, type sẽ được sử dụng trong việc lọc dữ liệu theo yêu cầu từ website quản trị khi xem lại kết quả phân tích, do vậy các trường này sẽ được đánh index

```
db.collection.createIndex( { hash: 1 } )
```

Tương tự cho các trường còn lại

2.2.3. Xây dựng thành phần trang web quản trị

Như được thiết kế từ trước, phần này có chức năng cung cấp giao diện cho người dùng tương tác, nhận yêu cầu phân tích mới, và giao diện hiển thị log, các thông tin kết quả sau khi phân tích.

2.2.3.1. Máy chủ website

Máy chủ website có nhiệm vụ cung cấp và đáp ứng các API cho phía client. Công nghệ được sử dụng ở đây là nền tảng Flask trên ngôn ngữ Python. Các API được cung cấp bao gồm:

/api/getFilter/<hash_value>

/api/getlog/

/api/search/

/api/static/<hash_value>

/api/submit/

Trong đó:

- GetFilter: lấy về toàn bộ các api_type, api_name, pid, tid để người dùng lọc ra các log đang quan tâm.
- Getlog: trả về log được lọc theo các yêu cầu phía client.
- Search: tìm các tệp đã được phân tích theo các trường như tên tệp, mã hash, md5, tên gói.
- StaticInfo: trả về các thông tin tĩnh được phân tích từ tệp Manifest.
- SubmitFile: cho phép client tải lên các tệp để phân tích.

GetFilter sẽ thống kê bộ lọc Log tương ứng bao gồm các “type” và tên API cụ thể theo từng “type”. Mã Python:

```
@api_get_filter.route("/api/getFilter/", defaults={'hash_value':
None})
@api_get_filter.route("/api/getFilter/<hash_value>",
methods=['GET'])
def get_filter(hash_value):
    cursor_ = DB.cltLog.aggregate([
        {
            "$match": {
                "hash": hash_value
            }
        },
        {
            "$group": {
                "_id": "$type",
                "apiName": {"$addToSet": "$apiName"}
            }
        }
    ])
    type_api = json.loads(dumps(cursor_))

    cursor_ = DB.cltLog.aggregate([
        {
            "$match": {
                "hash": hash_value
            }
        },
        {
            "$group": {
                "_id": "$pid",
                "tid": {"$addToSet": "$tid"}
            }
        }
    ])
    pid = json.loads(dumps(cursor_))

    data = {"type": type_api,
```

```

        "pid": pid}

    return json.dumps(data)

```

2.2.3.2. *Trình khách website*

Webclient được xây dựng với chức năng cung cấp giao diện cho người dùng, gọi lên webserver với các api cho trước. Webclient được xây dựng với công nghệ AngularJs cho kết quả là một website đơn trang, với thành phần khung cố định và các controller thay đổi theo từng chức năng. Trong AngularJs các controller được định nghĩa là các thành phần hoạt động độc lập, thực hiện một nghiệp vụ nào đó. Trong hệ thống này các controller được sử dụng như sau:

- StaticCtrl: Thực hiện lấy, hiển thị các thông tin tĩnh của file.
- ToolCtrl: thực hiện việc tìm kiếm các kết quả theo nhập vào của người dùng
- LogViewCtrl: quan trọng nhất, là controller thực hiện việc xử lý các thay đổi của bộ lọc, phân trang, hiển thị log lên giao diện cho người dùng quan sát, giới hạn số log trên một trang là 1000 dòng
- SubmitCtrl: thực hiện việc lấy các file và gửi về cho máy chủ để phân tích

Dữ liệu gửi qua lại giữa client và server đều theo định dạng Json, dữ liệu json được sử dụng và hỗ trợ rất tốt cả phía server sử dụng python và client sử dụng AngularJs

Tương tự đối với hàm thực hiện Filter trên server, phía Client thực hiện đoạn mã để gọi lên, cùng với các tham số để filter hoạt động. Mã javascript

```

function doFilter(page) {
    filter = {"hash": $routeParams.hash,
              "page": current_page
            };
    if(filter_type_condition != null && filter_type_condition !=
"all")
        filter["type"] = filter_type_condition;
    if(filter_apiName_condition != null &&
filter_apiName_condition != "all" )
        filter["apiName"] = filter_apiName_condition;
    if(filter_pid_condition != null && filter_pid_condition !=
"all")
        filter["pid"] = filter_pid_condition;
    if(filter_tid_condition != null && filter_tid_condition !=
"all")

```

```

        filter["tid"] = filter_tid_condition;
    console.log(filter)
    $http.post('/api/getlog/', filter)
    .then(function(response){
        res_data = response.data;
        console.log(res_data)
        if(res_data.status == "OK"){
            $scope.log_array = res_data.data.logs;
            $scope.currentPage = res_data.data.currentPage;
            $scope.totalPage = res_data.data.totalPage;
            total_page = res_data.data.totalPage;
            $scope.nextAble = false;
            $scope.preAble = false;
        }
        else
            console.log("Error: "+res_data.msg);
    })
}

```

CHƯƠNG 3. THỬ NGHIỆM THỆ THỐNG VÀ ĐÁNH GIÁ KẾT QUẢ

3.1. Mô hình thử nghiệm

3.1.1. Các thành phần

Hệ thống gồm các thành phần, và được cài đặt theo mô hình đã thiết kế bao gồm:

- Hệ thống đã xây dựng
- Máy thật chạy ubuntu 18.04
- Mẫu mã độc polyvideo

3.1.2. Mẫu mã độc sử dụng

Trong lần thử nghiệm này mẫu mã độc được sử dụng là mẫu “**polyvideo**” tên được đặt theo tên package. Đây là mẫu mã độc thực hiện nhiều hành vi bao gồm: lấy thông tin thiết bị, quảng cáo sai phạm, tự cài thêm cái phần mềm độc hại, tự đăng ký dịch vụ gia tăng đối với các nhà mạng bao gồm cả các nhà mạng di động tại Việt Nam. Mẫu này xuất hiện với nhiều biến thể khác nhau và hiện tại vẫn còn nhiều mẫu đang hoạt động và lan rộng, xuất hiện cả trên Google Play lẫn các kho ứng dụng thứ ba, đồng thời cũng phát tán thông qua các ứng dụng độc hại khác.

Thông tin mã độc:

Tên : polyvideo

Tên file: polyvideo_6122.apk

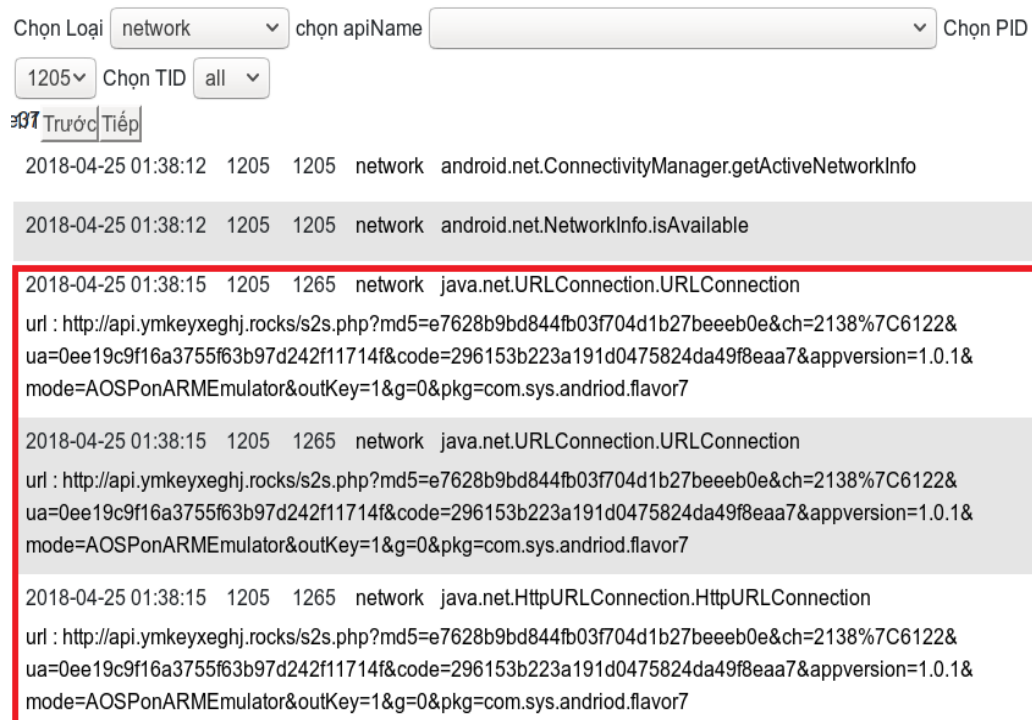
Package: com.sys.andriod.flavor7

Mã md5: d30221cfe0d1376c0ab3454a57305a2b

Mã sha: e39683fc82a5af6e581f1be779df6302d8409ac2b73b29b386492e373aa5382f

3.2. Phân tích kết quả phân tích của hệ thống

Kết quả đầu tiên ta quan tâm từ hệ thống là các địa chỉ website mà mã độc truy cập vào, các địa chỉ này có thể là địa chỉ C&C.



Hình 3.1 Các kết nối của mã độc

Hình 3.1 cho thấy kết quả của hệ thống đưa ra cho ta thấy một tên miền được mã độc kết nối đến với nhiều tham số theo phương thức GET, bao gồm thông tin của thiết bị. Trong thực tế phân tích với nhiều biến thể, dòng **polyvideo** kết nối tới nhiều C&C khác nhau nhưng giao thức vẫn giữ nguyên. Tiếp tục nhìn vào Log ta thấy.

Ngoài việc thực thi mã có sẵn, mã độc có thao tác với một file có tên 108classes.dex, ngay dưới đó là hàng loạt các hàm được sử dụng trong việc mã hóa. Có thể phỏng đoán rằng file 108classes.dex là file thực thi bổ sung và đã được mã hóa bằng thuật toán DES, với khóa được nhận ra thông qua log. Việc các hàm giải mã được gọi liên tục trước khi một phương thức nào đó được gọi một cách xen kẽ, ta có thể dự đoán được rằng mã độc này có sử dụng phương pháp gọi phương thức theo Java Reflection. Quan sát hình 3.2 để thấy điều này

```

2018-04-25 01:38:10 1205 1205 file java.io.File.File
path : /data/user/0/com.sys.andriod.flavor7/shared_prefs/dex.xml.bak

2018-04-25 01:38:10 1205 1219 file java.io.File.exists
path : /data/user/0/com.sys.andriod.flavor7/shared_prefs/dex.xml.bak

2018-04-25 01:38:10 1205 1219 file java.io.File.exists
path : /data/user/0/com.sys.andriod.flavor7/shared_prefs/dex.xml

2018-04-25 01:38:10 1205 1205 file java.io.File.File
path : /data/data/com.android.apache/108classes.dex

2018-04-25 01:38:10 1205 1205 file java.io.File.exists
path : /data/data/com.android.apache/108classes.dex

2018-04-25 01:38:10 1205 1205 crypto javax.crypto.spec.DESKeySpec.DESKeySpec
key : [71, 105, 101, 79, 98, 100, 107, 111]
offset : 0

2018-04-25 01:38:10 1205 1205 crypto javax.crypto.spec.DESKeySpec.DESKeySpec
key : [71, 105, 101, 79, 98, 100, 107, 111]

2018-04-25 01:38:10 1205 1205 crypto javax.crypto.SecretKeyFactory.getInstance
algorithm : DES

2018-04-25 01:38:10 1205 1205 crypto javax.crypto.spec.SecretKeySpec.SecretKeySpec
algorithm : DES
key : [71, 105, 101, 79, 98, 100, 107, 111]

2018-04-25 01:38:10 1205 1205 crypto javax.crypto.Cipher.getInstance

```

Hình 3.2 Mã độc thao tác với file dex bổ sung

```

2018-04-25 01:38:15 1205 1205 file java.io.File.File
path :

2018-04-25 01:38:15 1205 1205 file java.io.File.File
path :

2018-04-25 01:38:15 1205 1205 dexclassloader dalvik.system.DexClassLoader.DexClassLoader
dexPath : /data/data/com.android.apache/108classes.dex
optimizedDirectory : /data/user/0/com.sys.andriod.flavor7/app_temp
libraryPath : null

2018-04-25 01:38:15 1205 1205 file java.io.File.File
path : /data/user/0/com.sys.andriod.flavor7/shared_prefsMichael_S2SSDK_SPTAG.xml

2018-04-25 01:38:15 1205 1205 file java.io.File.File
path : /data/user/0/com.sys.andriod.flavor7/shared_prefsMichael_S2SSDK_SPTAG.xml

2018-04-25 01:38:15 1205 1205 file java.io.File.File
path : /data/user/0/com.sys.andriod.flavor7/shared_prefs/Michael_S2SSDK_SPTAG.xml.bak

2018-04-25 01:38:15 1205 1264 file java.io.File.exists
path : /data/user/0/com.sys.andriod.flavor7/shared_prefs/Michael_S2SSDK_SPTAG.xml.bak

2018-04-25 01:38:15 1205 1264 file java.io.File.exists
path : /data/user/0/com.sys.andriod.flavor7/shared_prefs/Michael_S2SSDK_SPTAG.xml

```

Hình 3.3 File dex được thực thi

Theo dõi tiếp log được thể hiện ở hình 3.3. Đến đây ta có thể khẳng định file 108classes.dex phía trên là file thực thi được mã độc nạp vào trong lúc thực thi.

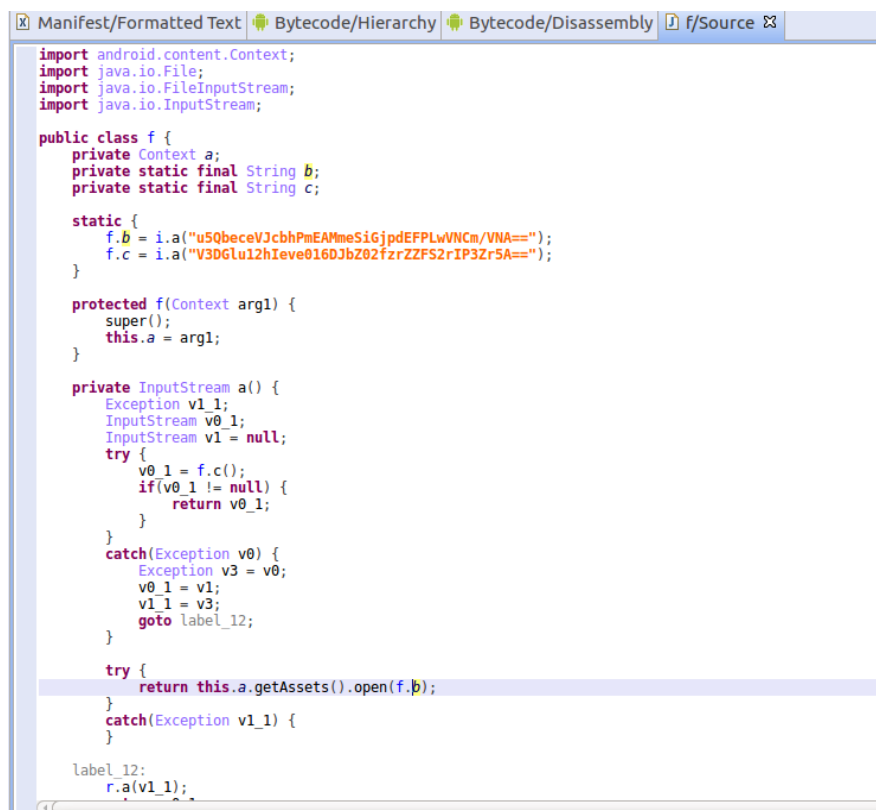
Ngoài ra mã độc còn thực hiện các hành vi khác như: lấy ID google của thiết bị, lấy thông tin về các package đã được cài đặt, thông tin độ phân giải màn hình, thường được các loại adware thực hiện. Mã độc cũng thực hiện việc tự ẩn khỏi danh sách ứng dụng và cài lên màn hình chính một biểu tượng để khởi động giả, dưới dạng một webview để mở trang web youtube.com

3.3. Đánh giá kết quả phân tích

3.3.1. So sánh với kết quả phân tích tĩnh

Để kiểm tra tính đúng đắn của kết quả từ hệ thống giám sát, kết quả phân tích sẽ được đối chiếu lại với mã thực thi của mã độc.

Mẫu **polyvideo** sử dụng phương pháp gọi phương thức theo **Java Reflection**, theo đó các phương thức không được gọi trực tiếp, tên các phương thức được mã hóa và được giải mã để gọi trong lúc thực thi. Một file được để trong assets của apk với tên blue.png chính là file được giải mã có tên **108classes.dex** và được gọi lên như kết quả của giám sát của hệ thống.



```
import android.content.Context;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;

public class f {
    private Context a;
    private static final String b;
    private static final String c;

    static {
        f.b = i.a("u5QbeceVJcbhPmEAMmeSigjpdEFPLwVNCm/VNA==");
        f.c = i.a("V3DGLu12hIeve0160JbZ02fzrZZFS2rIP3Zr5A==");
    }

    protected f(Context arg1) {
        super();
        this.a = arg1;
    }


    private InputStream a() {
        Exception v1_1;
        InputStream v0_1;
        InputStream v1 = null;
        try {
            v0_1 = f.c();
            if(v0_1 != null) {
                return v0_1;
            }
        } catch (Exception v0) {
            Exception v3 = v0;
            v0_1 = v1;
            v1_1 = v3;
            goto label_12;
        }

        try {
            return this.a.getAssets().open(f.b);
        } catch (Exception v1_1) {
        }

        label_12:
        r.a(v1_1);
    }
}
```

Hình 3.4 File blue.png được trích xuất

Một số chuỗi được sử dụng trong đó có URL mà hệ thống bắt được, trong file các chuỗi này đã được mã hóa, kết quả dưới đây là kết quả sau khi giải mã như `javax.crypto.spec.SecretKeySpec`. Trong các chuỗi trên còn có một số phương thức sẽ được gọi theo Java Reflection.

The image shows a screenshot of an IDE window with the title bar 'Manifest/Formatted Text | Bytecode/Hierarchy | Bytecode/Disassembly | String_Encode/Source'. The code is decompiled Java source code for a class named 'String_Encode'. It features a series of public static String variables labeled p through z, followed by a static block containing assignments for String_Encode.a through String_Encode.B. The assignments include various strings, URLs, and class names like 'javax.crypto.spec.SecretKeySpec' and 'javax.crypto.Cipher'.

```
public static String p;  
public static String q;  
public static String r;  
public static String s;  
public static String t;  
public static String u;  
public static String v;  
public static String w;  
public static String x;  
public static String y;  
public static String z;  
  
static {  
    String_Encode.a = "Michael_S2SSDK_SPTAG";  
    String_Encode.b = "S2SSDK_HasS2S";  
    String_Encode.c = "key";  
    String_Encode.d = "data";  
    String_Encode.e = "javax.crypto.spec.SecretKeySpec";  
    String_Encode.f = "AES";  
    String_Encode.g = "javax.crypto.Cipher";  
    String_Encode.h = "getInstance";  
    String_Encode.i = "java.security.Key";  
    String_Encode.j = "init";  
    String_Encode.k = "doFinal";  
    String_Encode.l = "http://api.ymkeyxeghj.rocks/s2s.php";  
    String_Encode.m = "http://api.ymkeyxeghj.rocks/statisticsNew.php";  
    String_Encode.n = "http://api.ymkeyxeghj.rocks/s2sDeductCH.php?ch=";  
    String_Encode.o = "AnalyticFYTech";  
    String_Encode.p = "AnalyticFYTech_Time";  
    String_Encode.q = "AnalyticFYTech_Index";  
    String_Encode.r = "AnalyticFYTech_Code";  
    String_Encode.s = "GET";  
    String_Encode.t = "POST";  
    String_Encode.u = "ser-Agent";  
    String_Encode.v = "Fiddler";  
    String_Encode.w = "Content-Type";  
    String_Encode.x = "application/json";  
    String_Encode.y = "tpp";  
    String_Encode.z = "0";  
    String_Encode.A = 3;  
    String_Encode.B = "/key";  
}
```

Hình 3.5 Một số chuỗi sau khi được giải mã

3.3.2. Độ chính xác của hệ thống

Các API được giám sát đầy log và đưa về hệ thống chính xác cùng với các tham số đầu vào trong lời gọi hàm, giúp người phân tích có được cái nhìn tổng quát cũng như kịch bản hoạt động của mã độc, định hướng phân tích và rút ngắn thời gian phân tích. Trong một số tình huống cần phản ứng nhanh với C&C, hệ thống giám sát có thể nhanh chóng đưa ra C&C điều khiển để ngăn chặn mã độc làm việc và lây nhiễm. Tuy nhiên, một số API chưa được giám sát có thể dẫn đến mất log, không

kiểm soát được. Việc này cần nhiều thời gian nghiên cứu, kinh nghiệm lập trình để hoàn thiện về sau.

Kết Luận

Hệ thống giám sát hành vi mã độc được thiết kế để giám sát, phân tích động các mã độc trên hệ điều hành Android. Sau quá trình nghiên cứu thì thực nghiệm đã thu được các kết quả khả quan, có ý nghĩa thực tế. Là công cụ hỗ trợ rất nhiều cho việc phân tích mã độc, rút ngắn thời gian giúp phản ứng nhanh khi có sự cố mã độc xảy ra. Việc phân tích tự động tuy chưa đủ chi tiết bằng việc phân tích tĩnh thủ công, vốn cần rất nhiều kiến thức và công sức của người thực hiện phân tích. Thay vào đó ta có được hệ thống có khả năng thực hiện nhanh, và khả năng phân tích nhiều mẫu cùng một lúc. Đây là một lợi thế rất lớn khi thực hiện phân tích nhiều mẫu thuộc cùng một dòng, khi đã biết rõ kỹ thuật được mã độc thực hiện và chỉ cần trích xuất các thông tin quan trọng cần quan tâm.

Phụ lục:

Danh sách các API được giám sát

<code>android.content.IntentFilter.addAction</code>	<code>android.net.ConnectivityManager.getMobileDataEnabled</code>
<code>java.io.File.File</code>	<code>android.net.ConnectivityManager.isDefaultNetworkActive</code>
<code>java.io.File.canExecute</code>	<code>android.net.ConnectivityManager.setGlobalProxy</code>
<code>java.io.File.canRead</code>	<code>android.net.ConnectivityManager.setAirplaneMode</code>
<code>java.io.File.canWrite</code>	<code>javax.net.ssl.HttpURLConnection.HttpURLConnection</code>
<code>java.io.File.delete</code>	<code>java.net.HttpURLConnection.HttpURLConnection</code>
<code>java.io.File.exists</code>	<code>android.net.NetworkInfo.getType</code>
<code>java.io.File.isHidden</code>	<code>android.net.NetworkInfo.setType</code>
<code>java.io.File.lastModified</code>	<code>android.net.NetworkInfo.getSubtype</code>
<code>java.io.File.setLastModified</code>	<code>android.net.NetworkInfo.setSubtype</code>
<code>javax.crypto.SecretKeyFactory.getInstance</code>	<code>android.net.NetworkInfo.getTypeName</code>
<code>javax.crypto.spec.DESKeySpec.DESKeySpec</code>	<code>android.net.NetworkInfo.getSubtypeName</code>
<code>javax.crypto.spec.IvParameterSpec.IvParameterSpec</code>	<code>android.net.NetworkInfo.isConnectedOrConnecting</code>
<code>javax.crypto.spec.SecretKeySpec.SecretKeySpec</code>	<code>android.net.NetworkInfo.isConnected</code>
<code>android.media.MediaRecorder.setOutputFile</code>	<code>android.net.NetworkInfo.isAvailable</code>
<code>android.os.PowerManager.newWakeLock</code>	<code>android.net.NetworkInfo.setIsAvailable</code>

<code>android.os.PowerManager.goToSleep</code>	<code>android.net.NetworkInfo.getState</code>
<code>android.os.PowerManager.wakeUp</code>	<code>java.net.URLConnection.URLConnection</code>
<code>android.os.PowerManager.isScreenOn</code>	<code>android.app.admin.DevicePolicyManager.lockNow</code>
<code>android.os.PowerManager.isInteractive</code>	<code>android.app.admin.DevicePolicyManager.wipeData</code>
<code>android.os.PowerManager.reboot</code>	<code>android.app.admin.DevicePolicyManager.installCaCert</code>
<code>android.os.PowerManager.isPowerSaveMode</code>	<code>java.lang.Runtime.exec</code>
<code>android.os.PowerManager.isDeviceIdleMode</code>	<code>java.lang.Runtime.exit</code>
<code>android.os.PowerManager.shutdown</code>	<code>android.app.ApplicationPackageManager.getPackageInfo</code>
<code>android.os.PowerManager.acquire</code>	<code>android.app.ApplicationPackageManager.getApplicationInfo</code>
<code>android.os.PowerManager.release</code>	<code>android.app.ApplicationPackageManager.getInstalledPackages</code>
<code>android.os.PowerManager.isHeld</code>	<code>android.app.ApplicationPackageManager.getInstalledApplications</code>
<code>android.net.wifi.WifiManager.getChannelList</code>	<code>android.app.ApplicationPackageManager.queryIntentActivities</code>
<code>android.net.wifi.WifiManager.isWifiScannerSupported</code>	<code>android.content.ContentResolver.query</code>
<code>android.net.wifi.WifiManager.startScan</code>	<code>android.app.ContextImpl.startActivity</code>
<code>android.net.wifi.WifiManager.startCustomizedScan</code>	<code>android.app.ContextImpl.startActivityAsUser</code>
<code>android.net.wifi.WifiManager.getWpsNfcConfigurationToken</code>	<code>android.app.ContextImpl.sendBroadcast</code>
<code>android.net.wifi.WifiManager.getConnectionInfo</code>	<code>android.app.ContextImpl.sendBroadcastMultiplePermissions</code>

android.net.wifi.WifiManager.getScanResults	android.app.ContextImpl.sendOrderedBroadcast
android.net.wifi.WifiManager.getCountryCode	android.app.ContextImpl.sendStickyBroadcast
android.net.wifi.WifiManager.getDhcpInfo	android.app.ContextImpl.startService
android.net.wifi.WifiManager.setWifiEnabled	android.app.ContextImpl.stopService
android.net.wifi.WifiManager.getWifiState	android.app.ContextImpl.bindService
android.net.wifi.WifiManager.isWifiEnabled	android.app.ContextImpl.getSystemService
android.net.wifi.WifiManager.calculateSignalLevel	dalvik.system.DexClassLoader.DexClassLoader
android.net.wifi.WifiManager.startWifi	android.telephony.SmsManager.sendMessage
android.net.wifi.WifiManager.stopWifi	android.telephony.SmsManager.sendMultipartTextMessage
android.net.wifi.WifiManager.connect	android.webkit.WebView.loadUrl
android.net.wifi.WifiManager.disable	android.webkit.WebView.postUrl
android.net.wifi.WifiManager.WifiLock.acquire	android.webkit.WebView.loadData
android.net.wifi.WifiManager.WifiLock.release	android.webkit.WebView.loadDataWithBaseUrl
android.net.wifi.WifiManager.WifiLock.isHeld	android.webkit.WebView.evaluateJavascript
android.net.wifi.WifiManager.createWifiLock	android.webkit.WebView.stopLoading
android.net.wifi.WifiManager.MulticastLock.acquire	android.webkit.WebView.reload
android.net.wifi.WifiManager.MulticastLock.release	android.webkit.WebView.canGoBack

<code>android.net.wifi.WifiManager. getCurrentNetwork</code>	<code>android.webkit.WebView.goBack</code>
<code>android.net.ConnectivityManager. getActiveNetworkInfo</code>	<code>android.webkit.WebView.canGoForward</code>
<code>android.net.ConnectivityManager. getActiveNetwork</code>	<code>android.webkit.WebView.goForward</code>
<code>android.net.ConnectivityManager. getActiveNetworkInfoForUid</code>	<code>android.webkit.WebView.canGoBackOrForward</code>
<code>android.net.ConnectivityManager. getNetworkInfo</code>	<code>android.webkit.WebView.goBackOrForward</code>
<code>android.net.ConnectivityManager. getAllNetworkInfo</code>	<code>android.webkit.WebView.clearView</code>
<code>android.net.ConnectivityManager. getNetworkForType</code>	<code>android.webkit.WebView.addJavaScriptInterface</code>
<code>android.net.ConnectivityManager. getAllNetworks</code>	<code>android.webkit.WebView.getSettings</code>
<code>android.net.ConnectivityManager. getActiveNetworkQuotaInfo</code>	<code>android.net.ConnectivityManager. getMobileDataEnabled</code>

Tài liệu tham khảo

- [1] John Wiley (2014), Android™ Hacker's Handbook.
- [2] Google, Android Security - <https://source.android.com/security/> - truy cập ngày 28/02/2018.
- [3] Google, The Android Source Code - <https://source.android.com/setup/> - truy cập ngày 28/02/2018.
- [4] Nikolay Elenkov (2015), Android Security Internals.