HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

ĐÀM TIẾN ĐẠT

PHƯƠNG PHÁP PHÁT HIỆN LỖ HỔNG MÃ NGUỒN DỰA TRÊN TẬP LUẬT

ĐỀ ÁN TỐT NGHIỆP THẠC SĨ KỸ THUẬT

(*Theo định hướng ứng dụng*)

HÀ NỘI - NĂM 2024

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

ĐÀM TIẾN ĐẠT

PHƯƠNG PHÁP PHÁT HIỆN LỖ HỔNG MÃ NGUỒN DỰA TRÊN TẬP LUẬT

Chuyên ngành: HỆ THỐNG THÔNG TIN
Mã số: 8.48.01.04

ĐỀ ÁN TỐT NGHIỆP THẠC SĨ KỸ THUẬT
(*Theo định hướng ứng dụng*)

NGƯỜI HƯỚNG DẪN KHOA HỌC :
PGS.TS. ĐỖ XUÂN CHỢ

HÀ NỘI - NĂM 2024

LỜI CAM ĐOAN

Tôi xin cam đoan đề án tốt nghiệp *"Phương pháp phát hiện lỗ hổng bảo mật dựa trên tập luật"* là công trình nghiên cứu thực sự của của riêng tôi.

Các số liệu, kết quả nêu trong đề án tốt nghiệp là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Hà Nội, tháng 02 năm 2024

Học viên

Đàm Tiến Đạt

LỜI CẢM ƠN

Lời đầu tiên, tôi xin được gửi lời biết ơn sâu sắc tới thầy giáo PGS. TS. Đỗ Xuân Chợ, thầy đã luôn khuyến khích, tận tình chỉ bảo, hướng dẫn và hỗ trợ tôi trong suốt quá trình nghiên cứu.

Tôi xin dành lời cảm ơn chân thành tới các thầy cô giáo của Học viện Công nghệ Bưu chính Viễn thông đã tận tình đào tạo, cung cấp cho tôi những kiến thức vô giá, tạo điều kiện tốt nhất cho tôi trong quá trình học tập, nghiên cứu.

Tôi xin chân thành cảm ơn.

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Kí hiệu	***Giải thích***	
	Tiếng Anh	***Tiếng Việt***
API	Application Programming Interface	Giao diện Lập trình ứng dụng
APK	Android Package	Định dạng tệp tin hệ điều hành Android
ATTT		An toàn thông tin
AV	Antivirus	Chống vi rút
CI/CD	Continuous Integration/Continuous Deployment	Tích hợp liên tục và triển khai liên tục
CNTT		Công nghệ thông tin
CVE	Common Vulnerabilities and Exposures	Các lỗ hổng bảo mật thông dụng
CVSS	Common Vulnerability Scoring System	Hệ thống chấm điểm lỗ hổng bảo mật phổ biến
HTTPS	Hypertext Transfer Protocol Secure	Giao thức truyền tải dữ liệu trên mạng internet
IDS	Intrusion Detection System	Hệ thống phát hiện xâm nhập
OWASP	Open Web Application Security Project	Dự án an toàn ứng dụng Web mở
SAST	Static Application Security Testing	Kiểm thử bảo mật ứng dụng tĩnh
UI	User Interfave	Giao diện người dùng

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH VẼ

MỞ ĐẦU

1. Lý do chọn đề tài:

Trong thời đại số hóa hiện nay, bảo mật trở thành một ưu tiên hàng đầu. Sự bất an về lỗ hổng mã nguồn có thể dẫn đến việc rò rỉ dữ liệu, mất cơ hội kinh doanh và ảnh hưởng đến uy tín của tổ chức. Nghiên cứu về phương pháp phát hiện lỗ hổng mã nguồn có thể giúp cải thiện bảo mật.

Dựa trên tập luật để phát hiện lỗ hổng mã nguồn có thể là một hướng tiếp cận đáng quan tâm, đặc biệt khi có sẵn nhiều kiến thức về lỗ hổng mã nguồn trong tập luật. Nghiên cứu về cách áp dụng tập luật để tự động phát hiện lỗ hổng mã nguồn có thể giúp giảm thiểu công sức thủ công trong việc kiểm tra mã nguồn.

Nghiên cứu về phương pháp phát hiện lỗ hổng mã nguồn dựa trên tập luật có thể đóng góp vào nguồn kiến thức về bảo mật phần mềm và có thể được chia sẻ với cộng đồng bảo mật để cải thiện phương pháp phát hiện và khắc phục lỗ hổng mã nguồn. Việc chọn đề tài ***"Phát hiện lỗ hổng mã nguồn dựa trên tập luật"*** thông qua đề tài này sẽ là giải pháp tối ưu để phát hiện các lỗ hổng mã nguồn từ đó giúp nâng cao an toàn ứng dụng và an toàn dữ liệu của các phần mềm.

2. Tổng quan về vấn đề nghiên cứu

Hiện nay nhiều phần mềm ứng dụng cho phép người dùng có thể truy cập và xem được các thông tin thông qua internet. Các phần mềm ứng dụng này hầu hết vẫn tồn tại các lỗ hổng và dễ bị khai thác trước các cuộc tấn công qua internet. Dựa vào việc sử dụng tập luật và công cụ để tìm ra các vấn đề tiềm ẩn trong mã nguồn của phần mềm, như các lỗ hổng bảo mật, sai sót lập trình, hoặc các vấn đề liên quan đến chuẩn mã hóa. Nếu phần mềm ứng dụng không được bảo vệ và khắc phục các lỗ hổng một cách thích hợp và chuẩn xác, kẻ tấn công có thể lợi dụng để xâm nhập vào hệ thống, đánh cắp, làm mất hay phá hủy cơ sở dữ liệu của hệ thống hoặc bản thân chính phần mềm.

3. Mục tiêu nghiên cứu của đề tài

Mục tiêu nghiên cứu của đề tài là tìm hiểu, nghiên cứu giải pháp, công cụ

để phát hiện các lỗ hổng của một phần mềm chứa các lỗi bảo mật.

4. Đối tượng và phạm vi nghiên cứu của đề tài

Đối tượng nghiên cứu của đề tài sẽ là công cụ hỗ trợ mã nguồn mở MobSF giúp phát hiện các lỗ hổng mã nguồn của phần mềm (cụ thể trong đề tài này là phần mềm Egov Quảng Nam phiên bản Android). Đề tài này tập trung nghiên cứu các phần sau:

- Nghiên cứu tổng quan về môi trường cài đặt phần mềm, các dạng tấn công và các lỗ hổng để khai thác và tấn công phổ biến.
- Tìm hiểu tập luật để rà quét và phát hiện lỗ hổng phần mềm.
- Nghiên cứu quá trình rà quét lỗ hổng bằng công cụ của một phần mềm

5. Phương pháp nghiên cứu của đề tài

- Phương pháp nghiên cứu: Kết hợp giữa lý thuyết và thực nghiệm thực tế rà quét và phát hiện các lỗ hổng mã nguồn và dựa trên tập luật bằng công cụ.

CHƯƠNG 1: TỔNG QUAN VỀ PHÁT HIỆN LỖ HỔNG MÃ NGUỒN DỰA TRÊN TẬP LUẬT

1.1. Tổng quan chung về lỗ hổng bảo mật và vấn đề phát hiện lỗ hổng bảo mật

1.1.1. Tổng quan về phát hiện và đánh giá lỗ hổng bảo mật

Tất cả các tài sản đều có giá trị quan trọng trong hoạt động của tổ chức và cần được bảo vệ. Do thông tin tồn tại và được lưu trữ dưới nhiều hình thức khác nhau, nên tổ chức phải có các biện pháp bảo vệ phù hợp để hạn chế rủi ro.

Bên cạnh những rủi ro về an toàn thông tin do bị tấn công phá hoại có chủ đích thông qua các lỗ hổng bảo mật, tổ chức cũng có thể gặp phải những rủi ro đối với thông tin, dữ liệu trong ứng dụng. Do đó, ngoài các biện pháp kỹ thuật hiện có, đơn vị phát triển cần nghiên cứu và áp dụng các công cụ phát hiện lỗ hổng phù hợp để giảm thiểu rủi ro.

Nguyên nhân gây ra lỗ hổng bao gồm:

- Độ phức tạp: Các hệ thống phức tạp làm tăng xác suất của lỗ hổng, sai sót trong cấu hình hoặc truy cập ngoài ý muốn.
- Tính phổ biến: Các loại mã, phần mềm, hệ điều hành và phần cứng có tính phổ biến sẽ làm tăng khả năng kẻ tấn công có thể tìm thấy hoặc có thông tin về các lỗ hổng đã biết.
- Mức độ kết nối: Thiết bị càng được kết nối nhiều thì khả năng xuất hiện lỗ hổng càng cao.
- Lỗi hệ điều hành: Giống như bất kỳ phần mềm nào khác, hệ điều hành cũng có thể có lỗ hổng. Các hệ điều hành không an toàn – chạy mặc định và để tất cả mọi người dùng có quyền truy cập đầy đủ sẽ có thể cho phép vi-rút và phần mềm độc hại thực thi các lệnh.
- Việc sử dụng Internet: Internet có rất nhiều loại phần mềm gián điệp và phần

mềm quảng cáo có thể được cài đặt tự động trên máy tính.

- Lỗi phần mềm: Lập trình viên có thể vô tình hoặc cố ý để lại một lỗi có thể khai thác trong phần mềm.
- Đầu vào của người dùng không được kiểm tra: Nếu trang web hoặc phần mềm cho rằng tất cả đầu vào đều an toàn, chúng có thể thực thi các lệnh SQL ngoài ý muốn.
- Coi thường bảo mật: chính vì sự coi thường này đã dẫn đến những sự thiệt hại hàng trăm triệu đô la trên toàn thế giới vì những cuộc tấn công của hacker. Doanh nghiệp/tổ chức nên quan tâm đến vấn đề này nhiều hơn, cập nhật xu thế bảo mật cũng như kiểu tấn công hiện nay là gì và đầu tư vào bảo vệ hệ thống.
- Sự lơ là, không phòng bị: có thể dẫn đến hậu quả khôn lường không chỉ riêng Việt Nam mà trên toàn thế giới, nhất là khoản phục hồi. Phải kiểm tra thường xuyên để phòng bị và sẵn sàng đối phó với những mối đe dọa từ những công cụ phần mềm quét bảo mật hoặc hệ thống giám sát được các nhà cung cấp uy tín khuyên dùng.
- Mở cửa cho sự gian lận: thời buổi hiện nay, những quy trình liên quan đến thông tin cá nhân của con người thì sẽ trở thành miếng mồi ngon cho lừa đảo. Nếu không có hệ thống giám sát thích hợp tại chỗ, các quy trình kinh doanh này có thể sẽ bị xâm phạm.
- An ninh di động/gia đình/di chuyển: bây giờ con người chúng ta có thể làm việc ở bất kỳ nơi đâu, ở nhà hoặc ngoài đường. Điều này có nghĩa là bất kỳ phương thức bảo mật mạng nào được triển khai trong tổ chức cũng đều phải được mở rộng ra ngoài phạm vi của văn phòng. Các thiết bị di động và máy tính xách tay cần phải được bảo mật và nhân viên phải được thông báo về các rủi ro và kế hoạch ứng phó.
- Cách xử lý sự cố: những cách xử lý sự cố kém hiệu quả sẽ dẫn tới thời gian hồi phục lâu, chi phí đội lên cao, mất niềm tin từ khách hàng.

- IoT(Internet of Things): internet vạn vật ngày nay đang dần trở nên phổ biến trong cuộc sống thông qua các thiết bị thông minh, kèm theo đó là những nỗi lo khi kẻ xấu lợi dụng chúng để tấn công hệ thống tài nguyên mạng doanh nghiệp.
- Đối tác/bên thứ ba: doanh nghiệp phải phối hợp với đối tác và bên cung cấp giải pháp phần mềm nhằm kiểm tra rà soát tất cả, tuyệt đối không để kẻ xấu lợi dụng lỗ hổng của đối tác để từ đó tấn công qua bên mình.
- Con người: Lỗ hổng lớn nhất trong bất kỳ tổ chức nào là con người đằng sau hệ thống đó. Hành vi của con người có thể dẫn đến bị lợi dụng và xâm phạm nếu không được giám sát. Chưa kể ở đây, một hành động của con người có thể dẫn đến sự xâm phạm an ninh doanh nghiệp như nhân viên có mưu đồ xấu xa đem bảo mật công ty tuồn ra ngoài hoặc cho kẻ khác đăng nhập vào hệ thống công ty, hoặc chỉ vì thiếu kiến thức mà có những hành động vô tình dẫn đến tai họa về sau. Tấn công phi kỹ thuật là mối đe dọa lớn nhất đối với đa số các tổ chức.

Rủi ro an ninh mạng thường được phân loại là một dạng lỗ hổng. Tuy nhiên, lỗ hổng (vulnerability) và rủi ro (risk) không giống nhau.

Cần hiểu rằng rủi ro là xác suất và tác động của việc một lỗ hổng bị khai thác. Nếu tác động và xác suất này là thấp thì có nghĩa là mức rủi ro thấp. Ngược lại, nếu tác động và xác suất này là cao nghĩa là mức rủi ro cao.

Công cụ phát hiện lỗ hổng giúp tổ chức, đơn vị phát triển thực hiện việc kiểm soát và định hướng cho các hoạt động đảm bảo ATTT. Việc phát hiện sớm các lỗ hổng bảo mật sẽ giúp công tác đảm bảo ATTT tại tổ chức được đảm bảo, được xem xét đánh giá định kỳ và không ngừng cải tiến để đối phó với các rủi ro mới phát sinh. Công cụ phát hiện lỗ hổng đảm bảo ATTT trong tổ chức sẽ giảm sự phụ thuộc vào cán bộ thực thi và luôn được xem xét, đánh giá để nâng cao hiệu quả.

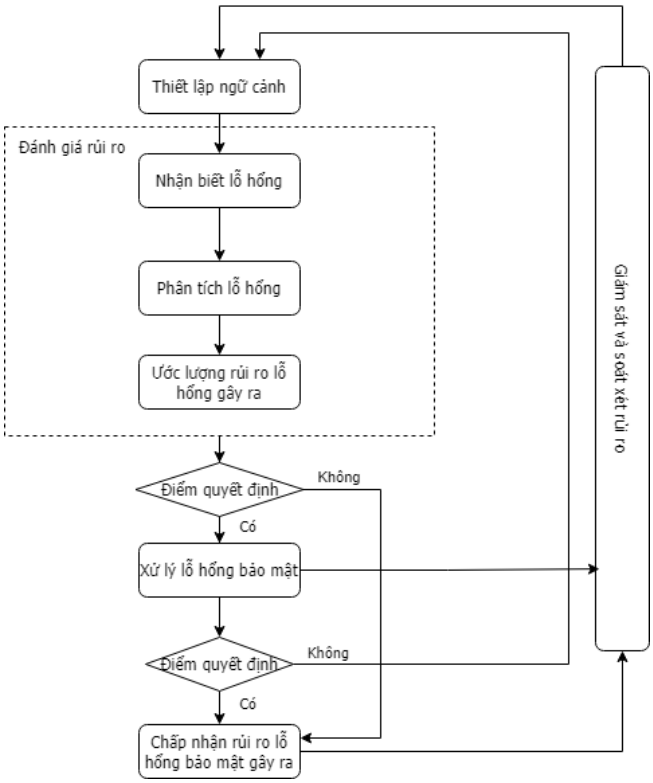
Việc tìm kiếm các lỗ hổng bảo mật trong một môi trường hệ thống nhất định đã trở nên dễ dàng hơn nhờ các công cụ khác nhau. Công cụ tự động được thiết kế

chuyên dung cho tìm kiếm lỗ hổng, điểm yếu, vấn đề cần giải quyết trong phần mềm hay ứng dụng.

1.1.2. Quy trình quản lý rủi ro

Khái niệm quản lý và xác định rủi ro các hệ thống của tổ chức được hiểu như là việc áp dụng các biện pháp xử lý, nhằm tiết giảm đầu tư vào nguồn lực, đảm bảo dự phòng, đánh giá và kiểm soát các rủi ro có thể. Với mục tiêu về các nghiệp vụ của tổ chức không bị ảnh hưởng, sai lệch. Kết quả mong muốn cho quản lý rủi ro là kiểm soát và điều hành tốt các hoạt động của tổ chức. Mọi thông tin và các hệ thống, quy trình, cán bộ liên quan đều là tài sản của tổ chức (Theo ISO/IEC 27001:2013).

Sơ đồ các luồng :



Hình 1.1. Sơ đồ thể hiện quy trình quản lý rủi ro

Luồng thực hiện quy trình:

Bước 1: Thiết lập ngữ cảnh, đầu vào, đầu ra và công cụ rà quét lỗ hổng.

Bước 2: Nhận biết lỗ hổng nhằm xác định nguy cơ, các mối đe dọa tiềm ẩn,

các điểm yếu đang tồn tại bên trong ứng dụng, có thể lợi dụng cho các mục đích xấu.

Bước 3: Phân tích rủi ro nhằm đánh giá tác động về an toàn bảo mật thông tin từ các nguy cơ đã nhận biết được ở Bước 2.

Bước 4: Ước lượng được rủi ro dựa trên các phân tích ở Bước 3 nhằm đánh giá mức độ rủi ro.

Bước 5: Từ các rủi ro đã phân tích thông qua các lỗ hổng đã phát hiện được xác định các lỗ hổng có nguy cơ cao để quyết định xử lý lỗ hổng. Nếu nguy cơ cao sẽ được yêu cầu khắc phục và đánh giá lại sau khi đã khắc phục xong còn các nguy cơ còn lại sẽ được khắc phục sau.

Bước 6: Sau khi đã xử lý lỗ hổng nguy cơ cao sẽ tiếp tục được đánh giá để quyết định có chấp nhận được các rủi ro lỗ hổng gây ra. Nếu chấp nhận được thì phần mềm, ứng dụng sẽ được thông qua còn nếu không sẽ quay lại Bước 1 để đánh giá lại.

Tất cả các bước từ thiết lập ngữ cảnh đến đánh giá rủi ro và kết luận chấp nhận rủi ro đều được giám sát và soát xét kỹ lưỡng nhằm đảm bảo sự chặt chẽ và an toàn cho cả quy trình.

Thiết lập ngữ cảnh bao gồm đầu vào, đầu ra, phương án thực hiện của ứng dụng cần đánh giá:

Đầu vào: Đơn vị phát triển chuẩn bị tập tin đóng gói cài đặt cho thiết bị Android (.apk) bản cuối để đánh giá rủi ro an toàn thông tin.

Đầu ra: Báo cáo thống kê các cảnh báo về lỗ hổng bảo mật và giải trình khắc phục của đơn vị phát triển.

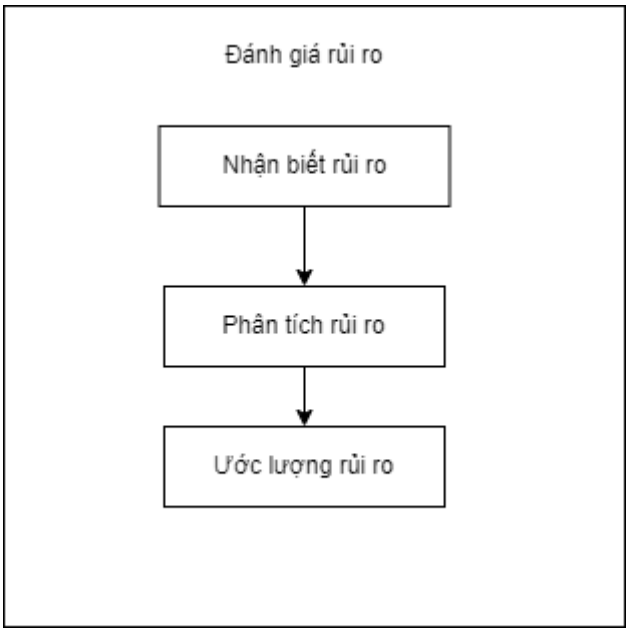
Phương án thực hiện: Thiết lập môi trường và công cụ rà quét lỗ hổng bảo mật sau đó từ báo cáo chi tiết của công cụ tổng hợp thành báo cáo các lỗ hổng cần khắc phục.

Nhằm xác định các nguy cơ có thể xảy ra với hệ thống, xác định các điểm yếu đang tồn tại, đánh giá mức độ tiềm ẩn, rủi ro có thể gặp phải. Từ đó, có thể phân loại và đánh giá mức độ theo thứ tự các rủi ro.

Đầu vào: Bao gồm tiêu chí, phạm vi, giới hạn thực hiện đánh giá rủi ro.

Hành động: Cần xác định định tính hoặc định lượng, sắp xếp các tiêu chí theo mức độ ưu tiên và các mục tiêu liên quan.

Đầu ra: Báo cáo tổng hợp danh sách những rủi ro và lỗ hổng bảo mật đã được sắp xếp theo thứ tự ưu tiên theo các tiêu chí. Trong quy trình đánh giá rủi ro, có ba hoạt động như sau:



Hình 1.2. Các bước đánh giá rủi ro

a) Bước 1: Nhận biết, xác định

Nhận biết rủi ro nhằm xác định nguy cơ, các mối đe dọa tiềm ẩn, các điểm yếu đang tồn tại bên trong ứng dụng, có thể lợi dụng cho các mục đích xấu. Từ đó xác định được phương thức, các mối đe dọa và tác động có thể ảnh hưởng thiệt hại đến tài sản, hệ thống của tổ chức.

Mục đích là xác định nguyên nhân có thể gây ra thiệt hại tiềm ẩn và hiểu được lí do, phương thức, thời điểm, không gian mà thiệt hại có thể xảy ra. Các hoạt động gồm:

b) Bước 2: Phân tích

Phân tích rủi ro nhằm xác định, đánh giá mức độ ảnh hưởng đến ứng dụng, xác định nguyên nhân và bản chất rủi ro. Xác định đánh giá tác động về an toàn bảo mật thông tin từ các nguy cơ đã dự báo từ trước. Từ đó đánh giá mức độ tác động đến

tổ chức hoặc tài sản của tổ chức.

c) Bước 3: Ước lượng

Ước lượng được căn cứ trên các phân tích nhằm xác định mức độ rủi ro, đánh giá mức độ rủi ro có nguy cao hay không.

1.1.3. Đánh giá rủi ro

Thông qua phân tích hiện trạng tại thời điểm đánh giá của ứng dụng bao gồm thông tin ứng dụng, lỗ hổng bảo mật tồn tại, các biện pháp an toàn bảo mật đang dùng, từ đó xác định rủi ro và đánh giá ảnh hưởng của rủi ro tạo nên cho hệ thống, đồng thời đưa ra giải pháp nhằm giảm thiểu rủi ro.

Mục đích của đánh giá rủi ro là thông báo cho những người ra quyết định và hỗ trợ các phản ứng rủi ro bằng cách xác định:

- Các lỗ hổng cả bên trong ứng dụng.
- Tác động (tức là gây hại) cho các tổ chức có thể xảy ra do khả năng đe dọa khai thác lỗ hổng.
- Khả năng gây hại sẽ xảy ra.

1.1.4. Tại sao phải đánh giá rủi ro lỗ hổng bảo mật

Các doanh nghiệp, tổ chức ở mọi quy mô, lĩnh vực sản xuất, cơ quan quản lý nhà nước… đều có nguy cơ về tấn công mạng, đánh cắp dữ liệu, mã hóa, lộ thông tin khách hàng hoặc thông tin nội bộ, dẫn đến phải đối mặt với những rủi ro phức tạp và lớn hơn. Không những thế, việc khắc phục hậu quả sẽ rất tốn kém về nhân lực, thời gian, tiền bạc và uy tín của đơn vị. Vì vậy, tại sao phải đánh giá rủi ro lỗ hổng bảo mật mã nguồn gây ra? Có 2 lý do chính như sau: Xác định mức độ an ninh của ứng dụng, phòng ngừa giảm thiểu rủi ro từ sớm và đảm bảo an toàn trong tương lai của phần mềm.

- Có đánh giá tổng quan toàn bộ phần mềm, bao gồm các thành phần về mã nguồn.
- Đưa ra được các giải pháp khắc phục, nâng cấp và cập nhật các bản nâng

cấp ứng dụng có nguy cơ.

- Xác định và đánh giá được các mức độ rủi ro về ứng dụng. Giúp đơn vị phát triển có thể quyết định nâng cấp trong tương lai.
- Ứng dụng hiện tại đã triển khai giải pháp đảm bảo an ninh nào, đánh giá tình hình và đề xuất triển khai nếu chưa đáp ứng được yêu cầu.
- Phân loại mức độ rủi ro của ứng dụng, từ đó xác định các mục tiêu chính cho việc phòng ngừa, giảm thiểu rủi ro trong tương lai.

1.2. Phương pháp phát hiện lỗ hổng dựa trên tập luật

1.2.1. Công cụ và kỹ thuật phát hiện lỗ hổng

Phát hiện lỗ hổng mã nguồn là một quá trình quan trọng để đảm bảo tính bảo mật của ứng dụng và hệ thống. Dưới đây là một số công cụ và kỹ thuật phổ biến để phát hiện lỗ hổng mã nguồn:

Các Công Cụ Phát Hiện Lỗ Hổng:

1. **SonarQube:** SonarQube là một công cụ mã nguồn mở giúp phát hiện lỗ hổng mã nguồn thông qua việc thực hiện kiểm tra tự động với nhiều ngôn ngữ lập trình.
    - Phân tích chất lượng mã nguồn: SonarQube phân tích mã nguồn bằng cách sử dụng các kỹ thuật phân tích tĩnh để xác định các vấn đề về chất lượng mã nguồn như code smells, lỗi và lỗ hổng bảo mật. Nó cũng đánh giá khả năng bảo trì mã và cung cấp điểm tổng chất lượng mã.
    - Ngôn ngữ hỗ trợ: SonarQube hỗ trợ một loạt ngôn ngữ lập trình, bao gồm Java, C#, JavaScript, TypeScript, Python, Ruby và nhiều ngôn ngữ khác. Điều này làm cho nó đa dạng và áp dụng được cho môi trường phát triển đa dạng.
    - Tích hợp với CI/CD: Nó tích hợp mượt mà với quy trình CI/CD liên tục. Điều này cho phép kiểm tra và đánh giá chất lượng mã nguồn tự động như một phần của quá trình xây dựng và triển khai liên tục.

- Bảng điều khiển và báo cáo: SonarQube cung cấp một bảng điều khiển trực tuyến hiển thị báo cáo chi tiết về các chỉ số chất lượng mã. Điều này bao gồm vấn đề được phát hiện, xu hướng chất lượng mã tổng thể và thông tin khác có liên quan.
- Quy tắc và chất lượng: Nó đi kèm với một bộ quy tắc được xác định trước cho mỗi ngôn ngữ lập trình được hỗ trợ. Những quy tắc này định nghĩa tiêu chuẩn viết mã và các thực hành tốt. Chất lượng có thể được cấu hình để đặt tiêu chí cho việc chấp nhận hoặc từ chối các thay đổi mã nguồn dựa trên điều kiện định trước.
- Phân tích bảo mật: SonarQube bao gồm các tính năng phân tích tập trung vào bảo mật để xác định và báo cáo các lỗ hổng bảo mật trong mã. Điều này giúp giải quyết các vấn đề về bảo mật sớm trong vòng đời phát triển.
- Cộng đồng và hệ sinh thái plugin: SonarQube có một cộng đồng tích cực và hỗ trợ một hệ thống plugin cho phép người dùng mở rộng chức năng của nó. Có nhiều plugin sẵn có để tích hợp với các công cụ khác và hỗ trợ các ngôn ngữ lập trình bổ sung.
- Tích hợp với IDE: Nó cung cấp các plugin cho các môi trường phát triển tích hợp phổ biến như Eclipse và IntelliJ IDEA, cho phép nhà phát triển thực hiện kiểm tra chất lượng mã trực tiếp trong môi trường phát triển của họ.
- Mã nguồn mở: SonarQube là mã nguồn mở, điều này có nghĩa là mã nguồn là miễn phí để kiểm tra và sửa đổi. Tính công khai của nền tảng khuyến khích sự đóng góp và cải tiến từ cộng đồng.

Các nhà phát triển và nhóm phát triển sử dụng SonarQube như một công cụ tích cực để duy trì chất lượng mã, xác định vấn đề sớm và đảm bảo rằng các dự án phần mềm tuân theo tiêu chuẩn viết mã.

2. **Checkmarx:** Checkmarx là một công cụ chuyên nghiệp được sử dụng để phân tích mã nguồn và phát hiện các lỗ hổng bảo mật.
    - Kiểm thử tĩnh mã nguồn: Checkmarx sử dụng kỹ thuật SAST để phân tích mã nguồn của ứng dụng mà không cần chạy ứng dụng. Điều này giúp phát hiện các lỗ hổng bảo mật và các vấn đề liên quan đến chất lượng mã nguồn sớm trong quá trình phát triển.
    - Phân tích bảo mật tự động: Checkmarx thực hiện phân tích tự động của mã nguồn để xác định các lỗ hổng bảo mật và tìm kiếm các mô hình tấn công potentional mà có thể được khai thác.
    - Hỗ trợ nhiều ngôn ngữ lập trình: Checkmarx hỗ trợ nhiều ngôn ngữ lập trình, bao gồm Java, C#, JavaScript, Python, và nhiều ngôn ngữ khác, làm cho nó linh hoạt và sử dụng được trong nhiều môi trường phát triển.
    - Integrations và Plugin: Công cụ này tích hợp tốt với nhiều công cụ phát triển phần mềm và quy trình CI/CD. Nó cung cấp plugin cho các môi trường phát triển phổ biến như IntelliJ IDEA, Visual Studio, và cũng tích hợp với các hệ thống quản lý mã nguồn như GitHub và GitLab.
    - Bảo mật ứng dụng toàn diện: Checkmarx không chỉ tập trung vào việc phát hiện lỗ hổng bảo mật mà còn hỗ trợ các tổ chức xây dựng và duy trì một chính sách bảo mật ứng dụng toàn diện.
    - Báo cáo và quản lý mối quan nguy hiểm: Nó cung cấp báo cáo chi tiết về các vấn đề bảo mật, đồng thời hỗ trợ quản lý mối quan nguy hiểm để đảm bảo rằng các vấn đề quan trọng được ưu tiên và giải quyết đúng cách.

Checkmarx giúp các tổ chức và nhóm phát triển tăng cường an toàn cho ứng dụng của họ thông qua quá trình kiểm thử bảo mật hiệu quả và tích hợp sâu rộng trong quy trình phát triển phần mềm.

3. **Fortify Static Code Analyzer:** Fortify của Micro Focus cung cấp một công cụ phân tích tĩnh mã nguồn để tìm kiếm và phát hiện các lỗ hổng bảo mật.
    - Kiểm thử tĩnh mã nguồn (SAST): Fortify SCA sử dụng phương pháp kiểm thử tĩnh để phân tích mã nguồn mà không cần chạy ứng dụng. Điều này giúp phát hiện lỗ hổng bảo mật và các vấn đề liên quan đến an toàn phần mềm sớm trong quá trình phát triển.
    - Hỗ trợ nhiều ngôn ngữ lập trình: Fortify hỗ trợ nhiều ngôn ngữ lập trình phổ biến, bao gồm Java, C#, C++, JavaScript, và nhiều ngôn ngữ khác. Điều này làm cho nó linh hoạt và sử dụng được trong nhiều môi trường phát triển.
    - Phân tích bảo mật tự động: Fortify SCA tự động phân tích mã nguồn để xác định và báo cáo về các lỗ hổng bảo mật, code smells, và các vấn đề khác có thể ảnh hưởng đến an toàn của ứng dụng.
    - Integrations với quy trình phát triển: Công cụ này có thể tích hợp với quy trình CI/CD (Continuous Integration/Continuous Deployment) và các công cụ quản lý mã nguồn như Git, SVN, và hỗ trợ tích hợp với các IDE phổ biến như Eclipse và Visual Studio.
    - Báo cáo và quản lý mối quan nguy hiểm: Fortify SCA cung cấp báo cáo chi tiết về các lỗ hổng bảo mật được phát hiện, giúp nhóm phát triển và an ninh đánh giá và ưu tiên các vấn đề để giải quyết.
    - Chấp Nhận Tự Động và Điều Chỉnh Linh Hoạt: Có khả năng tùy chỉnh và cấu hình để đảm bảo phương tiện kiểm thử linh hoạt và phản ánh chính sách bảo mật của tổ chức.

    Fortify SCA là một trong những công cụ SAST được sử dụng để đảm bảo an toàn và chất lượng mã nguồn trong quá trình phát triển ứng dụng.
4. **Veracode:** Veracode là một dịch vụ kiểm tra mã nguồn tự động được tích hợp vào chuỗi phát triển liên tục để phát hiện và báo cáo lỗ hổng mã nguồn.

- Kiểm Thử Bảo Mật Toàn Diện: Veracode cung cấp giải pháp kiểm thử bảo mật ứng dụng toàn diện, bao gồm cả kiểm thử tĩnh và động. Điều này giúp tổ chức phát hiện và giải quyết các vấn đề bảo mật từ quá trình phát triển đến triển khai.
- Kiểm Thử Tĩnh (SAST): Veracode Static Analysis sử dụng phương pháp kiểm thử tĩnh để phân tích mã nguồn của ứng dụng mà không cần chạy ứng dụng. Điều này giúp phát hiện lỗ hổng bảo mật và vấn đề liên quan đến chất lượng mã nguồn sớm trong quá trình phát triển.
- Kiểm Thử Động (DAST): Ngoài ra, Veracode cũng cung cấp kiểm thử động để phân tích ứng dụng trong môi trường chạy thực tế, giúp phát hiện các lỗ hổng có thể được tận dụng trong môi trường sản xuất.
- Hỗ Trợ Nhiều Ngôn Ngữ Lập Trình: Veracode hỗ trợ nhiều ngôn ngữ lập trình, bao gồm Java, C#, JavaScript, Python, Ruby, và nhiều ngôn ngữ khác, để đáp ứng đa dạng của ứng dụng phần mềm.
- Báo cáo và quản lý mối quan nguy hiểm: Cung cấp báo cáo chi tiết về các vấn đề bảo mật, đồng thời hỗ trợ quản lý mối quan nguy hiểm để đảm bảo rằng các vấn đề được ưu tiên và giải quyết đúng cách.
- Quy trình phát triển: Veracode tích hợp chặt chẽ với quy trình phát triển, CI/CD, và các công cụ quản lý mã nguồn như Jenkins, Jira, và nhiều công cụ khác.
- SaaS (Software as a Service): Veracode cung cấp dưới dạng dịch vụ, giúp giảm thiểu gánh nặng về quản lý hạ tầng và duy trì cập nhật phần mềm.

Veracode giúp tổ chức xây dựng và duy trì ứng dụng an toàn và chất lượng bằng cách tập trung vào kiểm thử bảo mật trong toàn bộ quy trình phát triển phần mềm.

5. **Brakeman:** Brakeman là một công cụ dành cho ứng dụng Ruby on Rails để phát hiện lỗ hổng bảo mật trong mã nguồn Ruby.
    - Kiểm thử tĩnh mã nguồn (SAST): Brakeman sử dụng phương pháp kiểm thử tĩnh để phân tích mã nguồn của ứng dụng Ruby on Rails mà không cần chạy ứng dụng. Điều này giúp phát hiện các lỗ hổng bảo mật và vấn đề liên quan đến chất lượng mã nguồn sớm trong quá trình phát triển.
    - Hỗ trợ ruby on rails: Brakeman được thiết kế đặc biệt để kiểm thử bảo mật cho ứng dụng Ruby on Rails, một framework phổ biến cho việc phát triển web trong ngôn ngữ lập trình Ruby.
    - Phát hiện các loại tấn công phổ biến: Brakeman tập trung vào việc phát hiện các loại tấn công phổ biến như SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), và nhiều vấn đề bảo mật khác liên quan đến ứng dụng Ruby on Rails.
    - Báo cáo chi tiết: Công cụ cung cấp báo cáo chi tiết về các lỗ hổng bảo mật được phát hiện, bao gồm thông tin chi tiết về vấn đề, đường dẫn, và lời khuyên để sửa lỗi.
    - Tích hợp với quy trình phát triển: Brakeman có thể tích hợp vào quy trình phát triển và hệ thống CI/CD để thực hiện kiểm thử tự động trong quy trình triển khai.
    - Mã nguồn mở: Brakeman là một dự án mã nguồn mở, có nghĩa là mã nguồn của nó có sẵn và có thể được cộng đồng sử dụng và đóng góp.

    Brakeman là một công cụ hữu ích cho các nhà phát triển và nhóm phát triển sử dụng Ruby on Rails để giúp đảm bảo rằng ứng dụng của họ được triển khai với mức độ an toàn và bảo mật cao.
6. **Bandit:** Bandit là một công cụ kiểm tra lỗ hổng bảo mật trong mã nguồn Python, tập trung vào việc tìm kiếm lỗ hổng liên quan đến an ninh mã nguồn.

- Kiểm thử tĩnh mã nguồn (SAST): Bandit sử dụng phương pháp kiểm thử tĩnh để phân tích mã nguồn Python mà không cần chạy ứng dụng. Điều này giúp phát hiện các lỗ hổng bảo mật và vấn đề an ninh từ mã nguồn Python sớm trong quá trình phát triển.
- Phát hiện các loại tấn công phổ biến: Bandit tập trung vào việc phát hiện các loại tấn công phổ biến trong mã nguồn Python, bao gồm SQL injection, cross-site scripting (XSS), code injection, và nhiều vấn đề bảo mật khác.
- Báo cáo chi tiết: Công cụ cung cấp báo cáo chi tiết về các lỗ hổng bảo mật được phát hiện, bao gồm thông tin chi tiết về vấn đề, dòng mã nguồn, và lời khuyên để sửa lỗi.
- Duy trì đa dạng python: Bandit hỗ trợ nhiều phiên bản của Python và được duy trì để đảm bảo tính tương thích và hiệu suất trong các dự án sử dụng các phiên bản Python khác nhau.
- Tích hợp với quy trình phát triển: Bandit có thể tích hợp vào quy trình phát triển và hệ thống CI/CD để thực hiện kiểm thử tự động trong quy trình triển khai.
- Mã nguồn mở: Bandit là một dự án mã nguồn mở, cho phép cộng đồng sử dụng và đóng góp vào phát triển và cải tiến của nó.

Bandit là một công cụ hữu ích để kiểm thử bảo mật mã nguồn Python và giúp đảm bảo rằng các dự án Python được triển khai với mức độ an toàn và bảo mật cao.

7. **OWASP Dependency-Check:** OWASP Dependency-Check kiểm tra các phụ thuộc của ứng dụng để phát hiện các thư viện đã được công bố về lỗ hổng bảo mật.
    - Phân tích các phần mềm phụ thuộc: Dependency-Check tập trung vào việc phân tích các phần mềm phụ thuộc của một dự án, như thư viện và framework, để xác định các lỗ hổng bảo mật có thể tồn tại trong các phiên bản cụ thể của các phần mềm này.

- Nền tảng đa dạng: Dependency-Check có thể chạy trên nhiều nền tảng khác nhau, bao gồm Windows, macOS, và Linux. Nó cũng có thể tích hợp vào quy trình CI/CD.
- Hỗ trợ nhiều ngôn ngữ và frameworks: Công cụ này hỗ trợ nhiều ngôn ngữ lập trình và frameworks phổ biến, chẳng hạn như Java, .NET, Python, Ruby, Node.js, và nhiều hơn nữa.
- Tích hợp với quy trình phát triển: Dependency-Check có thể tích hợp vào quy trình phát triển và hệ thống CI/CD để tự động thực hiện kiểm thử an ninh trong quá trình xây dựng và triển khai.
- Báo cáo chi tiết: Công cụ cung cấp báo cáo chi tiết về các lỗ hổng bảo mật được phát hiện, giúp nhóm phát triển hiểu rõ về nguy cơ và có thể đưa ra các biện pháp sửa lỗi cụ thể.
- Tương thích với nhiều định dạng báo cáo: Dependency-Check có thể xuất kết quả kiểm thử dưới nhiều định dạng khác nhau, bao gồm HTML, JSON, JUnit XML, và SARIF.
- Mã nguồn mở: OWASP Dependency-Check là một dự án mã nguồn mở, có nghĩa là cộng đồng có thể đóng góp vào phát triển và cải thiện của nó.

Dependency-Check giúp các nhà phát triển và quản trị an ninh thông tin xác định và giảm thiểu rủi ro an ninh liên quan đến các phần mềm phụ thuộc, giúp bảo vệ ứng dụng khỏi các lỗ hổng bảo mật có thể được tận dụng thông qua các thành phần sử dụng.

Kỹ Thuật Phát Hiện Lỗ Hổng Mã Nguồn:

- Kiểm tra tĩnh (Static Analysis): Sử dụng công cụ để kiểm tra mã nguồn mà không thực hiện thực thi chương trình. Các công cụ này tìm kiếm lỗ hổng từ mã nguồn trước khi chạy ứng dụng.
- Kiểm tra động (Dynamic Analysis): Thực hiện kiểm tra lỗ hổng trong mã nguồn trong quá trình chạy ứng dụng. Các kỹ thuật này thường bao gồm penetration testing và fuzz testing.

- Kiểm tra tự động (Automated Testing): Sử dụng kịch bản kiểm thử tự động để kiểm tra mã nguồn một cách định kỳ và tự động, giúp phát hiện lỗ hổng một cách nhanh chóng.
- Kiểm tra bảo mật phần mềm bên thứ ba: Sử dụng dịch vụ của các công ty chuyên về bảo mật phần mềm để kiểm tra mã nguồn và đưa ra báo cáo chi tiết về các lỗ hổng có thể xảy ra.
- Kiểm tra quy mô lớn (Large-Scale Analysis): Sử dụng công nghệ big data và machine learning để phân tích lỗ hổng mã nguồn trên quy mô lớn, giúp xác định các xu hướng và mô hình bảo mật.

Việc sử dụng một sự kết hợp giữa các công cụ và kỹ thuật này thường là lựa chọn tốt nhất để đảm bảo mức độ bảo mật cao cho mã nguồn. Mỗi công cụ có những ưu điểm và hạn chế riêng, và việc kết hợp sử dụng nhiều công cụ có thể giúp đảm bảo tính toàn vẹn và an toàn của ứng dụng Android.

1.2.2. Mô tả và phân loại tập luật

Tập luật là tập hợp các quy tắc mà hệ thống hoặc mạng so sánh để quyết định nên làm gì hoặc hành động nào được chấp thuận.

Ví dụ:

- Tập luật tường lửa: cho phép hoặc từ chối một gói dữ liệu khi đi qua.
- Tập luật IDS: cảnh báo khi có xâm nhập bất hợp pháp tới hệ thống.
- Tập luật AV: xóa file khi xác định có chứa mã độc.

Rà soát đánh giá phần mềm nhằm:

- Tìm ra những lỗi hoặc lỗ hổng bảo mật: lỗ hổng bảo mật, các vi phạm chính sách.
- Tìm ra các thiếu sót hoặc không hiệu quả làm ảnh hưởng đến hiệu suất của phần mềm.

Tập luật để phát hiện lỗ hổng mã nguồn phần mềm thường được thiết kế để kiểm tra mã nguồn của phần mềm và xác định các vấn đề bảo mật, lỗi lập trình, và các tiêu chí chất lượng mã khác. Dưới đây là một số chi tiết về các quy tắc thường

được áp dụng để phát hiện lỗ hổng mã nguồn:

1. Quy tắc về Bảo mật:

- Kiểm tra xác thực và ủy quyền: Phát hiện mã nguồn không an toàn liên quan đến xác thực và ủy quyền.
- Kiểm tra xâm phạm bảo mật: Phát hiện các lỗ hổng bảo mật phổ biến như SQL injection, Cross-Site Scripting (XSS), và các kỹ thuật tấn công khác.

2. Quy tắc về Hiệu suất:

- Kiểm tra lợi dụng tài nguyên: Phát hiện mã nguồn có thể gây ra lợi dụng tài nguyên không cần thiết hoặc tiêu tốn quá mức.

3. Quy tắc về Độ tin cậy:

- Phòng tránh chống mã độc hại: Kiểm tra mã nguồn để phát hiện và ngăn chặn việc tích hợp mã độc hại.

4. Quy tắc về Kiểm soát biên:

- Kiểm tra kiểm soát biên: Phát hiện các lỗ hổng liên quan đến kiểm soát biên không an toàn.

5. Quy tắc về Tiêu chuẩn mã:

- Kiểm tra định dạng mã: Đảm bảo rằng mã nguồn tuân theo các quy tắc về định dạng để dễ đọc và duy trì.
- Kiểm tra sự hiện diện của mã chết: Loại bỏ đoạn mã không sử dụng để giảm kích thước của ứng dụng.

6. Quy tắc về Ngôn ngữ và API:

- Kiểm tra sử dụng API nhạy cảm: Đảm bảo rằng các API nhạy cảm được sử dụng đúng cách và an toàn.

7. Quy tắc về Giao diện người dùng (UI):

- Kiểm tra sử dụng tài nguyên giao diện: Đảm bảo rằng tài nguyên giao diện được sử dụng đúng cách và không có lỗi.

Các quy tắc có thể được tinh chỉnh tùy thuộc vào yêu cầu cụ thể của dự án và loại ứng dụng.

Tập luật áp dụng thông qua một mô tả chung danh sách các lựa chọn gồm điều kiện và mô tả của điều kiện đó. Nếu không có điều kiện nào được đáp ứng thì ứng dụng không được cấp quyền thực thi yêu cầu, hành động tương ứng. Sau đây là trường hợp cụ thể:

message: Encryption Of Sensitive Application Data: là tiêu đề cho một tính năng hoặc cải tiến trong ứng dụng liên quan đến việc mã hóa dữ liệu nhạy cảm của ứng dụng.

selection: 'The application {} in non-volatile memory.': Đây là một phần của mẫu lựa chọn, mô tả một tình huống hoặc điều kiện liên quan đến việc lưu trữ ứng dụng trong bộ nhớ không tưởng. Mô tả về ứng dụng, được đại diện bằng {}.

choice: Phần này chứa một danh sách các lựa chọn cụ thể và mô tả của chúng. Mỗi lựa chọn bao gồm một điều kiện và mô tả tương ứng của điều kiện đó.

androidx\.security\.crypto\.(?:EncryptedFile|EncryptedSharedPreferences): Điều kiện đầu tiên, dựa trên biểu thức chính quy, có thể xác định xem ứng dụng có sử dụng các tính năng mã hóa dữ liệu nhạy cảm được cung cấp sẵn trong thư viện androidx.security.crypto không.

javax\.crypto\.Cipher|net\.sqlcipher\.|\.encryptionKey\(: Điều kiện thứ hai, cũng dựa trên biểu thức chính quy, xác định xem ứng dụng có sử dụng các phương pháp mã hóa dữ liệu nhạy cảm bằng cách triển khai Cipher trong gói javax.crypto, hoặc sử dụng thư viện net.sqlcipher, hoặc sử dụng phương thức encryptionKey() không.

else: does not encrypt files: Đây là phần khác, định nghĩa hành động nếu không có điều kiện nào trong danh sách được đáp ứng. Trong trường hợp này, nếu ứng dụng không sử dụng bất kỳ cơ chế mã hóa dữ liệu nhạy cảm nào được đề cập trong các điều kiện trước đó, nó sẽ không mã hóa các tập tin.

class: Security Functional Requirements là một phân loại hoặc nhóm của các yêu cầu chức năng liên quan đến bảo mật. Mô tả các lựa chọn được liên kết với yêu cầu chức năng bảo mật.

1.2.3. Khó khăn trong phát hiện lỗ hổng bảo mật

Với sự phát triển liên tục về công nghệ thông tin trên thế giới đã giúp các tổ chức, đơn vị giảm thiểu được sức người trong hầu hết các lĩnh vực. Nhưng bên cạnh đó, các tổ chức tội phạm công nghệ cao ngày càng tinh vi và nguy hiểm khi liên tục sử dụng công nghệ cao để sử dụng cho mục đích phá hoại, đánh cắp thông tin, gián điệp... Đòi hỏi các đơn vị cần phải tập trung đầu tư nhiều vào công tác đảm bảo an toàn thông tin của các phần mềm và ứng dụng. Để thực hiện được điều đó, cần phải thực hiện công tác thường xuyên dò quét, xác định các nguy cơ tiềm ẩn để có biện pháp khắc phục, tăng cường an toàn thông tin hệ thống và mã nguồn của các phần mềm, ứng dụng. Vì vậy, các tổ chức đơn vị cần phải có những đầu tư chi phí cho phát hiện các lỗ hổng bảo mật cho hệ thống của mình, trong đó bao gồm các chi phí về: Phần mềm, con người, quy trình ... những thách thức không nhỏ đối với các đơn vị khi gặp khó khăn trong việc lựa chọn nhà cung cấp về giải pháp đảm bảo an ninh hệ thống. Trong khi đó các tội phạm công nghệ cao ngày càng tinh vi, đòi hỏi các giải pháp dò quét cũng phải chủ động trong việc xác định các nguy cơ tiềm ẩn, hỗ trợ người đứng đầu đơn vị có thể ra những quyết định, chính sách phù hợp. Công nghệ không ngừng phát triển, việc quản lý rủi ro an toàn thông tin luôn gặp nhiều khó khăn vì các nguy cơ luôn có thể xảy ra, điều tốt nhất các đơn vị phát triển cần phải có những đầu tư nghiêm túc phù hợp với tình hình thực tế.

1.3. Kết luận chương 1

Các cuộc tấn công mạng, tội phạm công nghệ cao, gián điệp không ngừng gia tăng nhằm mục đích phá hoại, lấy cắp dữ liệu, bí mật của tổ chức doanh nghiệp và cả của nhà nước. Chúng có thể gây ra những hậu quả vô cùng nghiêm trọng đến các đơn vị, tổ chức ở mọi vùng miền, quốc gia, lãnh thổ trên thế giới. Trong thời gian tới, dự báo về tinh hình an ninh thông tin có những diễn biến khó lường, đặc biệt là các tội phạm công nghệ cao sử dụng trí tuệ nhân tạo trong lây nhiễm mã độc, tấn công với mục đích xấu nhằm đánh cắp, mã hóa thông tin. Vì vậy, để có thể phòng ngừa, giảm thiểu rủi ro, đòi hỏi các tổ chức doanh nghiệp phải quan tâm đến công tác đầu tư các hệ thống dò quét lỗ hổng bảo mật dùng cho nhận biết, đánh giá, phân loại… Từ đó có thể có thể phát hiện các lỗ hổng, điểm yếu của ứng dụng, kết xuất báo cáo đánh giá chi tiết và mức độ nghiêm trọng phục vụ công tác đảm bảo an toàn bảo mật thông tin. Trên các cơ sở lý thuyết đó, chương tiếp theo sẽ trình bày chi tiết về bài toán đặt ra và hướng giải quyết.

CHƯƠNG 2: PHƯƠNG PHÁP PHÁT HIỆN LỖ HỔNG MÃ NGUỒN DỰA TRÊN TẬP LUẬT

2.1. Tổng quan về công cụ phát hiện lỗ hổng mã nguồn

Công cụ phát hiện lỗ hổng mã nguồn là các ứng dụng được thiết kế để kiểm tra mã nguồn của phần mềm để xác định và báo cáo về các lỗ hổng bảo mật, tiềm ẩn, hoặc các vấn đề khác có thể ảnh hưởng đến tính toàn vẹn và an toàn của ứng dụng. Dưới đây là một tổng quan về công cụ phát hiện lỗ hổng mã nguồn:

Chức Năng Cơ Bản:

Phân Tích Tĩnh:

Kiểm tra lỗ hổng bảo mật: Các công cụ này kiểm tra mã nguồn để phát hiện và báo cáo về các lỗ hổng bảo mật như SQL injection, cross-site scripting (XSS), tràn bộ đệm, và nhiều vấn đề khác.

Phân tích quy ẩn: Các công cụ có thể phân tích các luồng điều khiển, dữ liệu và cấu trúc của ứng dụng để xác định lỗ hổng tiềm ẩn mà không cần chạy ứng dụng.

Tích Hợp Trong Quy Trình Phát Triển:

Tích hợp với ide: Nhiều công cụ có thể tích hợp trực tiếp vào môi trường phát triển tích hợp (IDE) như Eclipse, Visual Studio, hoặc IntelliJ IDEA để hỗ trợ lập trình viên phát hiện lỗ hổng ngay từ khi viết mã.

Tích hợp trong quy trình CI/CD: Các công cụ thường có khả năng tích hợp vào các hệ thống liên tục tích hợp và triển khai (CI/CD) để kiểm tra mã nguồn trong quy trình tự động.

Loại Công Cụ:

Tự Động Hóa:

Static Application Security Testing (SAST): Các công cụ SAST kiểm tra mã nguồn mà không chạy ứng dụng. Chúng phân tích tập tin mã nguồn và báo cáo về lỗ hổng mà chúng tìm thấy.

Dynamic Application Security Testing (DAST): Ngược lại với SAST, DAST chạy ứng dụng để phân tích lỗ hổng trong môi trường thực tế. Nó kiểm tra ứng dụng đang chạy và tìm kiếm lỗ hổng từ cấp độ runtime.

Ưu Điểm:

- Phát Hiện Sớm: Cung cấp khả năng phát hiện sớm các lỗ hổng khi vẫn ở cấp độ mã nguồn.
- Tự Động Hóa: Các công cụ thường có thể tự động hóa việc kiểm tra mã nguồn, giảm áp lực cho nhóm phát triển.
- Tích Hợp Liên Tục: Các công cụ thích hợp với quy trình liên tục tích hợp, giúp duy trì mã nguồn an toàn và bảo mật trong quá trình phát triển.

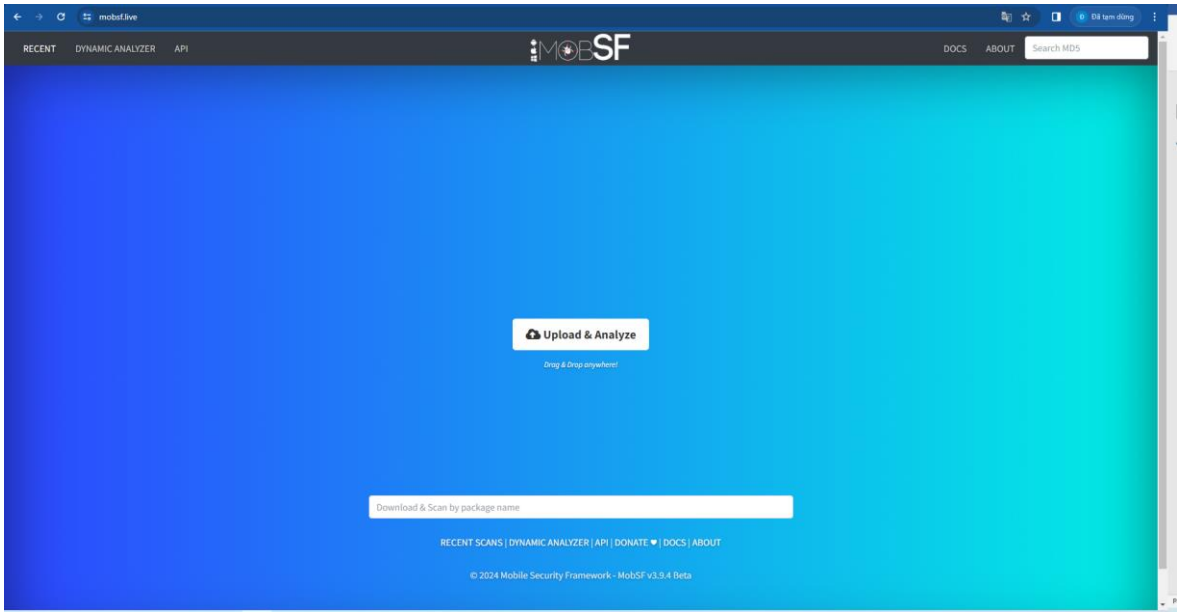
Nhược Điểm:

- False Positives: Có thể xuất hiện các kết quả giả mạo, làm tăng công việc kiểm tra và giảm độ chính xác.
- Phức Tạp Cho Người Mới: Sử dụng chúng có thể đòi hỏi sự hiểu biết sâu rộng về bảo mật và mã nguồn.
- Khả Năng Chấp Nhận: Các công cụ thường có khả năng chấp nhận tốt khi mã nguồn chưa được viết hoặc thiết kế đúng cách.

Việc sử dụng công cụ phát hiện lỗ hổng mã nguồn là một phần quan trọng trong chiến lược bảo mật phần mềm để đảm bảo rằng mã nguồn được viết và duy trì một cách an toàn và bảo mật.

2.2. Nghiên cứu về công cụ MobSF

MobSF là một công cụ mã nguồn mở được phát triển bởi Ajin Abraham được sử dụng để phân tích tự động APK. Đây là tập hợp các công cụ chạy dưới một giao diện, thực hiện các tác vụ riêng lẻ như Jadx, apktool, v.v. và hiển thị kết quả dưới một giao diện chung. Các báo cáo này cũng có thể được tải xuống ở định dạng PDF và phân tích chi tiết cùng với các ảnh chụp màn hình[1, tr.1].



Hình 2.1. Giao diện MobSF

2.2.1. Đánh giá MobSF với một số công cụ khác

Các công cụ rà quét lỗ hổng mã nguồn, bao gồm cả MobSF, thường có một số điểm chung và khác nhau. Dưới đây là một số điểm chung và khác nhau giữa MobSF và các công cụ rà quét lỗ hổng mã nguồn khác:

Điểm chung:

Phát hiện lỗ hổng bảo mật: Cả MobSF và các công cụ rà quét lỗ hổng mã nguồn khác đều được thiết kế để phát hiện các lỗ hổng bảo mật trong mã nguồn của ứng dụng, như lỗ hổng về quyền truy cập, mã độc, lỗ hổng mã nguồn mở, và các vấn đề khác liên quan đến bảo mật.

Hỗ trợ nhiều ngôn ngữ lập trình: Cả MobSF và các công cụ khác thường hỗ trợ nhiều ngôn ngữ lập trình phổ biến, bao gồm Java, Python, JavaScript, C/C++, và các ngôn ngữ khác. Một số công cụ chỉ hỗ trợ một loại ngôn ngữ lập trình như công cụ Brakeman chỉ hỗ trợ ngôn ngữ Ruby

Tính linh hoạt và mở rộng: Các công cụ này đều có tính linh hoạt và có thể được mở rộng thông qua các plugin hoặc tích hợp với các công cụ khác để tạo ra quy trình kiểm thử bảo mật tự động.

Điểm khác nhau:

Phạm vi hỗ trợ: MobSF thường chủ yếu tập trung vào ứng dụng di động (Android và iOS), trong khi các công cụ khác có thể hỗ trợ phân tích mã nguồn cho các loại ứng dụng khác nhau như web, desktop, và các hệ thống nhúng.

Tính năng chuyên sâu: Mỗi công cụ có các tính năng chuyên sâu riêng biệt, chẳng hạn như MobSF tập trung vào các vấn đề bảo mật cụ thể cho ứng dụng di động, trong khi các công cụ khác có thể tập trung vào kiểm tra chất lượng mã, phân tích mã nguồn mở, và nhiều hơn nữa.

Cộng đồng và hỗ trợ: Mức độ hỗ trợ từ cộng đồng và sự phát triển của các công cụ có thể khác nhau. Một số công cụ có cộng đồng lớn và tích cực hơn, trong khi các công cụ khác có thể có ít hỗ trợ hơn.

Giấy phép và chi phí: Các công cụ có thể được phân phối dưới các giấy phép khác nhau, bao gồm các phiên bản miễn phí và phiên bản trả phí với các tính năng mở rộng và hỗ trợ cao cấp hơn.

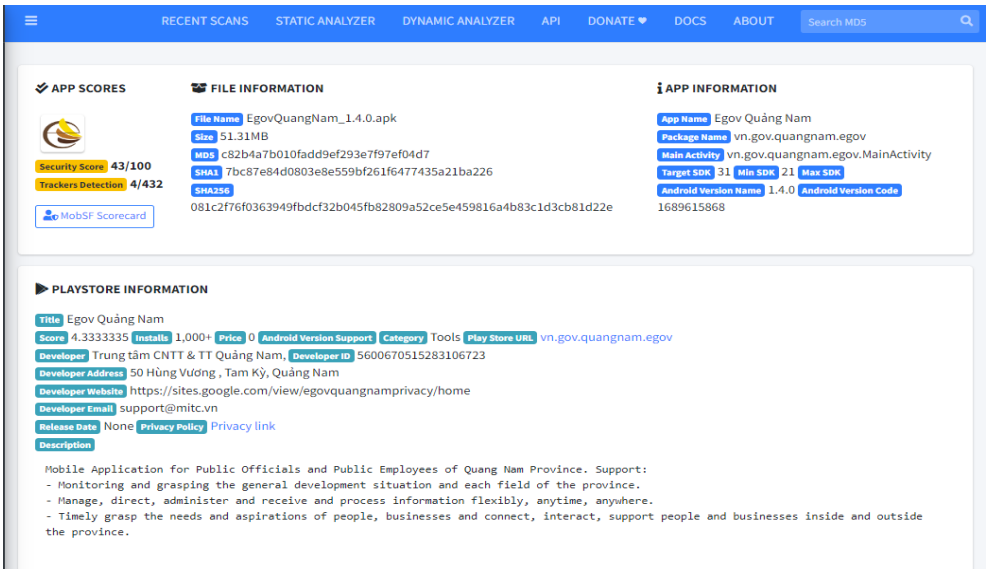
Ngữ cảnh đầu vào: Đối với MobSF là tệp APK để rà quét còn các công cụ khác sẽ tích hợp trong các công cụ phát triển ứng dụng để rà quét trực tiếp mã nguồn trong quá trình phát triển.

Tóm lại, MobSF và các công cụ rà quét lỗ hổng mã nguồn khác có nhiều điểm chung trong việc phát hiện lỗ hổng bảo mật, nhưng cũng có những khác biệt quan trọng về phạm vi hỗ trợ, tính năng, cộng đồng, và giấy phép. Lựa chọn giữa chúng phụ thuộc vào nhu cầu cụ thể của dự án và mục tiêu bảo mật. MobSF thường được sử dụng cho mục đích kiểm tra bảo mật cá nhân hoặc nhóm nhỏ hơn, và có thể yêu cầu kiến thức chuyên sâu hơn về bảo mật ứng dụng di động.

2.2.2. Báo cáo và các kết quả rà quét của công cụ MobSF

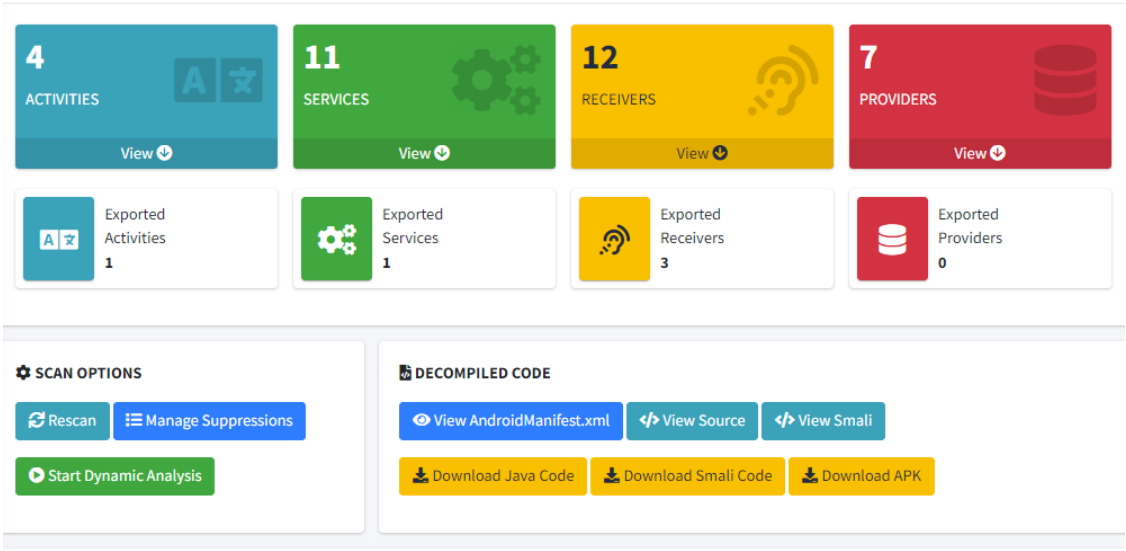
Sau khi thực hiện tiến trình phân tích tĩnh APK, trên trang đích các điểm mức độ nghiêm trọng được hiển thị. Tiếp theo, hàm băm, tên tệp và kích thước của APK

cũng được cung cấp. Trong cột thứ ba ở hàng đầu tiên là tên gói, hoạt động chính, phiên bản SDK tối thiểu và cả phiên bản ứng dụng. Mô tả của ứng dụng cũng được đưa ra.



Hình 2.2. Mô tả thông tin ứng dụng

- Trong các thẻ nhỏ, chúng ta thấy các thành phần ứng dụng khác nhau
- Tùy chọn để xem mã dịch ngược. Đây là mã được tạo bởi apktool, tệp tài nguyên cũng được giải mã. Và có thể dễ dàng phân tách và xem mã nguồn trong các lớp java riêng biệt.



Hình 2.3. Thành phần ứng dụng

Phân tích chứng chỉ người ký trong cột chứng chỉ, chúng ta có thể thấy chứng chỉ người ký, nơi người ta có thể tìm thấy thông tin quan trọng về nhà phát triển, quốc gia, tiểu bang, loại bí danh, kích thước bit, v.v.

Hình 2.4. Chứng chỉ người ký

Quyền ứng dụng có nhiều quyền khác nhau được phân loại là nguy hiểm hoặc bình thường. Theo quan điểm của nhà phân tích bảo mật, điều quan trọng là phải hiểu quyền nào có thể dẫn đến thiệt hại.

Ví dụ: nếu một ứng dụng có quyền truy cập vào phương tiện bên ngoài và lưu trữ thông tin quan trọng trên phương tiện bên ngoài thì có thể bị xem là nguy hiểm vì các tệp được lưu trữ trên phương tiện bên ngoài có thể đọc và ghi được trên toàn cục.

PERMISSION	STATUS	INFO	DESCRIPTION	CODE MAPPINGS
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.	Show Files
android.permission.ACCESS_WIFI_STATE	normal	view Wi-Fi status	Allows an application to view the information about the status of Wi-Fi.	Show Files
android.permission.CAMERA	dangerous	take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.	Show Files
android.permission.DOWNLOAD_WITHOUT_NOTIFICATION	unknown	Unknown permission	Unknown permission from android reference	
android.permission.FOREGROUND_SERVICE	normal	enables regular apps to use Service.startForeground.	Allows a regular application to use Service.startForeground.	Show Files
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.	Show Files
android.permission.READ_CALENDAR	dangerous	read calendar events	Allows an application to read all of the calendar events stored on your phone. Malicious applications can use this	Show Files

Hình 2.5. Quyền ứng dụng

Hoạt động có thể duyệt và phân tích an ninh mạng: Trong phần bảo mật mạng, ta có thể tìm thấy một số chi tiết về các vấn đề an ninh mạng liên quan đến ứng dụng. Những vấn đề này đôi khi có thể dẫn đến các cuộc tấn công nghiêm trọng như MiTM.

BROWSABLE ACTIVITIES

Search:

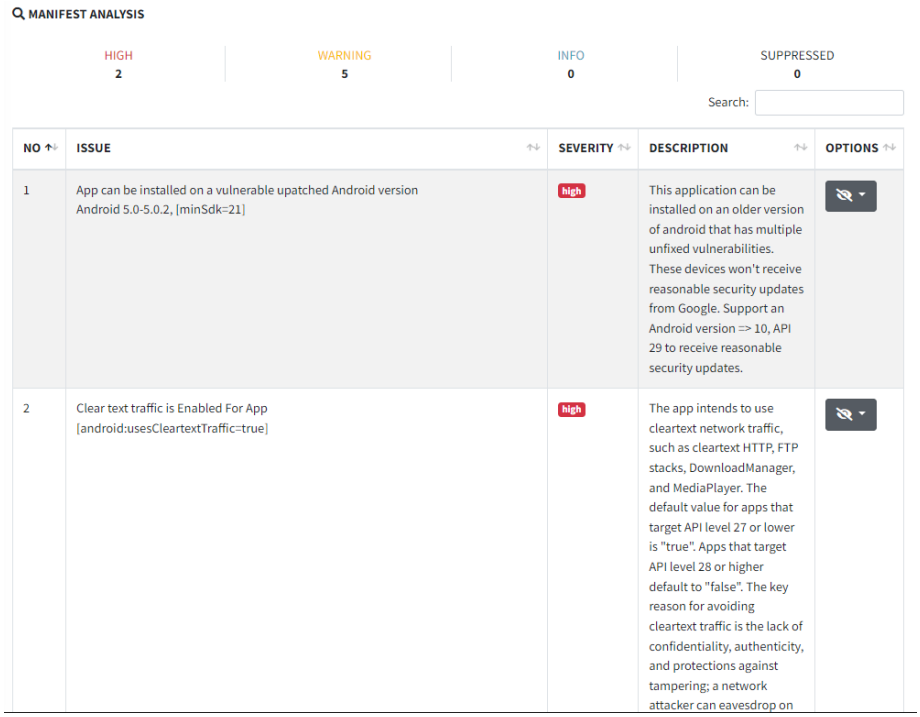
ACTIVITY	INTENT
vn.gov.quangnam.egov.MainActivity	**Schemes**: egovquangnam://,

Showing 1 to 1 of 1 entries

Previous 1 Next

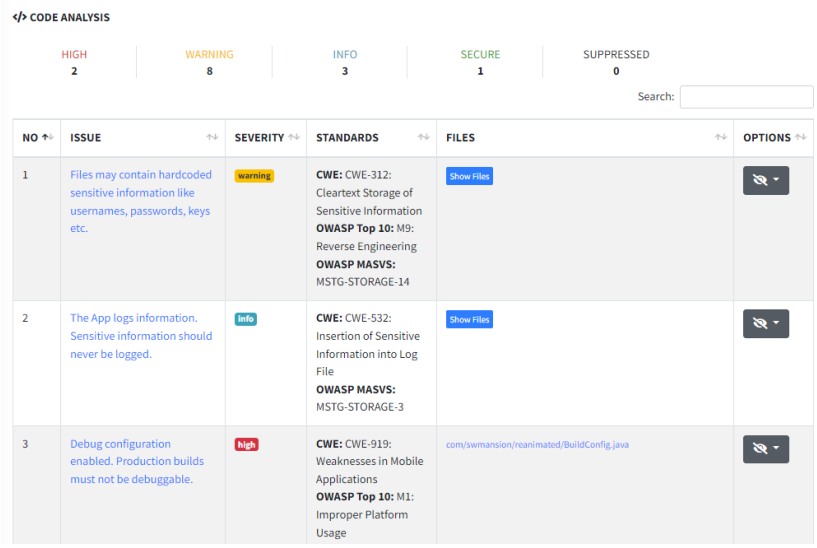
Hình 2.6. Hoạt động có thể duyệt

Phân tích Tệp kê khai: có thể tìm thấy nhiều thông tin từ tệp kê khai android như hoạt động nào được xuất, ứng dụng có thể gỡ lỗi hay không, lược đồ liệu, v.v. Để tham khảo, hãy xem ảnh chụp màn hình bên dưới.



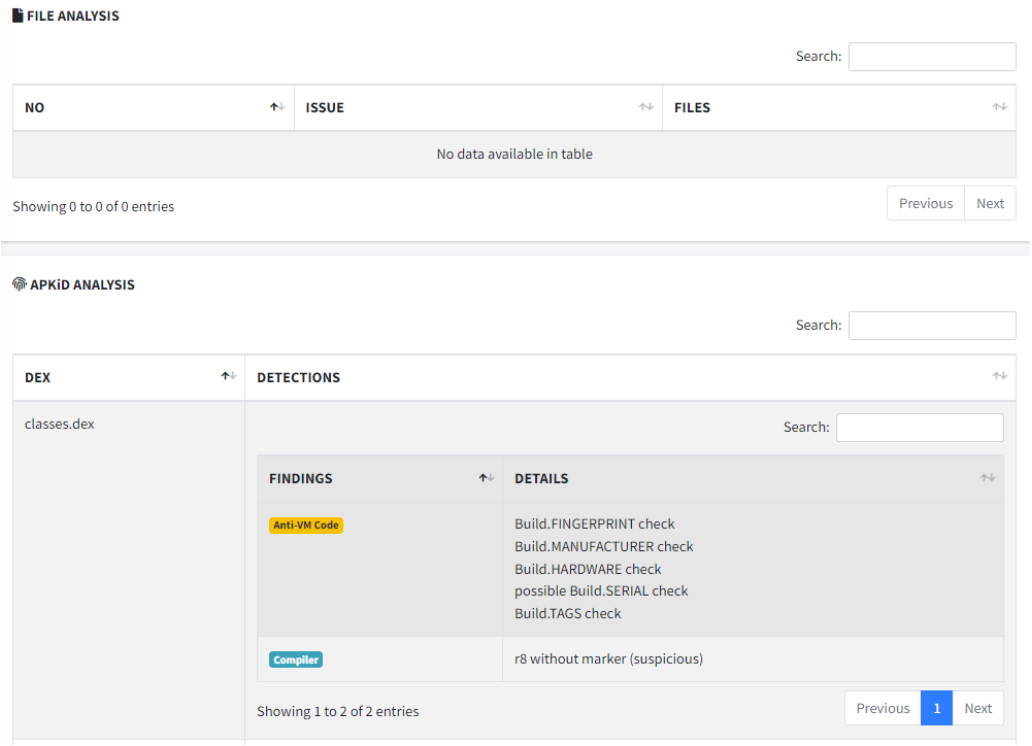
Hình 2.7. Phân tích tệp kê khai

Phân tích mã: có thể thấy rằng MobSF đã phân tích và so sánh một số hành vi của ứng dụng dựa trên các thông lệ tiêu chuẩn bảo mật của ngành như OWASP MSTG và lập bản đồ các lỗ hổng với OWASP Top 10. Các phân tích và so sánh này có thể giúp phân tích các tình huống khác nhau và tạo báo cáo dễ dàng hơn.



Hình 2.8. Phân tích mã

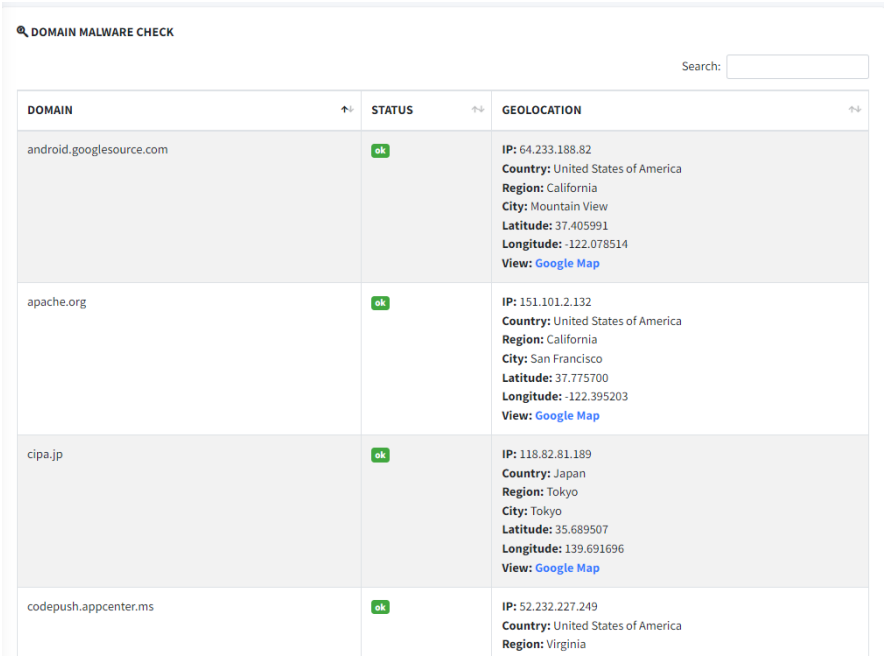
Phân tích phần mềm độc hại MobSF có tích hợp APKiD là một công cụ mã nguồn mở rất hữu ích để xác định các trình đóng gói, trình biên dịch, obfuscators khác nhau, v.v. trong các tệp android tương tự như PEiD trong APK.



Hình 2.9. Phân tích phần mềm độc hại

MobSF cũng trích xuất tất cả các URL / địa chỉ IP được mã hóa cứng hoặc đang được sử dụng trong ứng dụng và hiển thị trạng thái phần mềm độc hại cũng như

sử dụng ip2location để cung cấp vị trí địa lý IP.



DOMAIN MALWARE CHECK

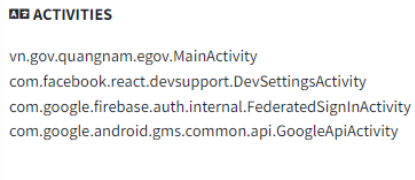
Search:

DOMAIN	STATUS	GEOLOCATION
android.googlesource.com	ok	**IP:** 64.233.188.82 **Country:** United States of America **Region:** California **City:** Mountain View **Latitude:** 37.405991 **Longitude:** -122.078514 **View:** Google Map
apache.org	ok	**IP:** 151.101.2.132 **Country:** United States of America **Region:** California **City:** San Francisco **Latitude:** 37.775700 **Longitude:** -122.395203 **View:** Google Map
cipa.jp	ok	**IP:** 118.82.81.189 **Country:** Japan **Region:** Tokyo **City:** Tokyo **Latitude:** 35.689507 **Longitude:** 139.691696 **View:** Google Map
codepush.appcenter.ms	ok	**IP:** 52.232.227.249 **Country:** United States of America **Region:** Virginia

Hình 2.10. URL/IP được trích xuất

Các email được mã hóa cứng trong MobSF cũng được hiển thị tại trang báo cáo. Tất cả điều này được thực hiện bằng cách sử dụng mã nguồn được dịch ngược. Thông thường, một pentester có thể tìm thấy các ID email quan trọng đang được sử dụng làm thông tin đăng nhập trên trang web của bên thứ ba để truy cập cơ sở dữ liệu. Như email thì các URL cũng thường được mã hóa cứng. Có thể tìm thấy các URL hấp dẫn đôi khi được sử dụng. Thông thường, các nhà phân tích nhận thấy các URL độc hại cũng được truy cập hoặc thậm chí là máy chủ C&C.

Các thành phần hoạt động hiện tại: Danh sách tất cả các hoạt động hiện có cũng có thể xem bằng MobSF. Điều này cung cấp thông tin chi tiết về khung của APK android. Ngoài ra, đôi khi jadx thay thế tên thật của lớp bằng một số chữ cái ngẫu nhiên nếu nhà phát triển đã áp dụng phương pháp xáo trộn, MobSF cũng có thể liên kết tên thật của nó[8, tr.50].

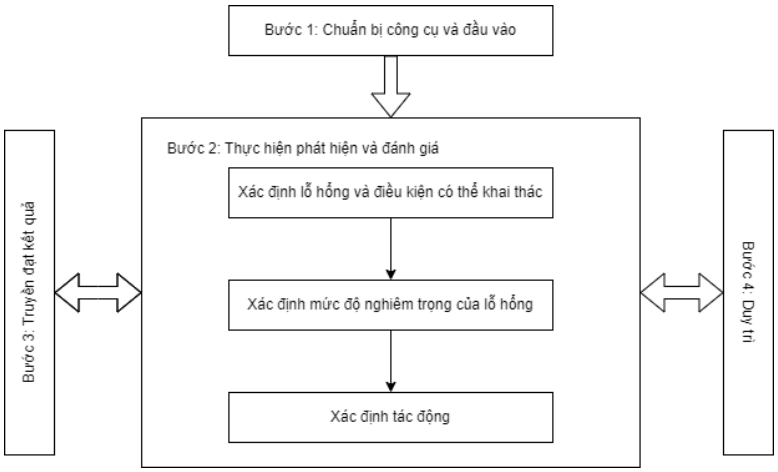


Hình 2.11. Các thành phần hoạt động

2.3. Quy trình phát hiện lỗ hổng mã nguồn dựa trên công cụ MobSF và tập luật

Thông qua phân tích hiện trạng của ứng dụng bao gồm mô tả thông tin ứng dụng, lỗ hổng bảo mật tồn tại, các biện pháp an toàn bảo mật đang dùng, từ đó xác định rủi ro và đánh giá ảnh hưởng của rủi ro tạo nên cho ứng dụng, đồng thời đưa ra giải pháp nhằm giảm thiểu rủi ro.

Quy trình phát hiện lỗ hổng mã nguồn bao gồm 4 bước, mỗi bước được chia thành một nhóm các nhiệm vụ. Đối với mỗi nhiệm vụ, hướng dẫn bổ sung cung cấp thông tin bổ sung cho thực hiện đánh giá rủi ro. Hình sau minh họa các bước cơ bản trong quy trình đánh giá rủi ro và các nhiệm vụ cụ thể để thực hiện đánh giá.



Hình 2.12. Các bước cơ bản trong quy trình đánh giá rủi ro

Bảng dưới đây là tổng hợp các nhiệm vụ đánh giá rủi ro:

Bảng 2.1. Bảng tổng hợp các nhiệm vụ phát hiện và đánh giá lỗ hổng

Nhiệm vụ	Mô tả nhiệm vụ
Bước 1: Chuẩn bị đánh giá	
Nhiệm vụ 1-1: Xác định mục đích	Xác định mục đích của phát hiện lỗ hổng theo thông tin mà đánh giá được dự định và đưa ra các quyết định.
Nhiệm vụ 1-2: Xác định phạm vi	Xác định phạm vi đánh giá rủi ro, khung thời gian được hỗ trợ và cân nhắc về công nghệ.
Nhiệm vụ 1-3: Xác định các đánh giá và hạn chế	Xác định các giả định và ràng buộc cụ thể theo đó đánh giá rủi ro được thực hiện.
Nhiệm vụ 1-4: Xác định nguồn thông tin	Xác định các nguồn thông tin mô tả, đe dọa, dễ bị tổn thương và tác động sẽ được sử dụng trong đánh giá rủi ro lỗ hổng có thể gây ra.
Bước 2: Thực hiện đánh giá	
Nhiệm vụ 2-1: Xác định nguồn gốc	Xác định và mô tả các nguồn đe dọa, bao gồm khả năng, ý định và đặc điểm mục tiêu cho các mối đe dọa đối nghịch và phạm vi ảnh hưởng đối với các mối đe dọa không đối nghịch.
Nhiệm vụ 2-2: Xác định sự kiện	Xác định các sự kiện đe dọa tiềm ẩn, mức độ liên quan của các sự kiện và các nguồn đe dọa có thể bắt đầu các sự kiện.
Nhiệm vụ 2-3: Xác định nhiệm	Xác định các lỗ hổng và các điều kiện có xu hướng ảnh hưởng đến khả năng các mối quan tâm đe dọa dẫn đến các

vụ và điều kiện bảo đảm	tác động bất lợi.
Nhiệm vụ 2-4: Xác định các tác động	Xác định các tác động bất lợi từ các sự kiện đe dọa đáng quan tâm, xem xét: (i) các đặc điểm của các nguồn đe dọa có thể bắt đầu các sự kiện; (ii) các lỗ hổng / điều kiện có khuynh hướng được xác định; và (iii) các biện pháp bảo vệ / biện pháp đối phó được lên kế hoạch hoặc thực hiện để cản trở các sự kiện đó.
Nhiệm vụ 2-5: Xác định rủi ro	Xác định rủi ro cho tổ chức từ các sự kiện đe dọa cần quan tâm: (i) tác động sẽ xảy ra từ các sự kiện; và (ii) khả năng xảy ra sự kiện.
Bước 3: Truyền đạt kết quả	
Nhiệm vụ 3-1: Truyền đạt kết quả	Truyền đạt kết quả cho những người ra quyết định tổ chức để hỗ trợ các phản ứng rủi ro.
Nhiệm vụ 3-2: Chia sẻ thông tin liên quan	Chia sẻ thông tin liên quan đến rủi ro được tạo ra trong quá trình đánh giá rủi ro với nhân viên tổ chức phù hợp.
Bước 4: Duy trì	
Nhiệm vụ 4-1: Giám sát các yếu tố rủi ro	Tiến hành giám sát liên tục các yếu tố rủi ro góp phần thay đổi rủi ro đối với hoạt động của ứng dụng.
Nhiệm vụ 4-2: Cập nhật đánh giá	Cập nhật đánh giá rủi ro hiện có.

Quy trình phát hiện lỗ hổng mã nguồn:

Bước 1: Thu thập thông tin

Thu thập tất cả thông tin liên quan đến mục tiêu cần đánh giá: Source code của

ứng dụng. Như xác định ngôn ngữ lập trình, Framework, … được sử dụng.

Bước 2: Xác định các điểm yếu/lỗ hổng bảo mật

Chuyên gia sẽ sử dụng những sản phẩm dò quét hàng đầu trên thế giới như: Fortify Static Code Analyzer,… để tìm kiếm, xác định các điểm yếu như:

• Dò quét các điểm yếu cấu hình.

• Dò quét các điểm yếu về xác thực.

• Dò quét các điểm yếu quản lý phiên.

• Dò quét các điểm yếu về ủy quyền.

• Dò quét điểm yếu liên quan đến dữ liệu đầu vào.

• Dò quét các điểm yếu mã hóa.

• Dò quét điểm yếu khả năng kiểm soát lỗi.

• Dò quét các điểm yếu liên quan đến đầu ra dữ liệu.

Bước 3: Tổng hợp kết quả và khuyến nghị khắc phục

Tổng hợp lại tất các các thông tin thu thập được, nội dung của quá trình làm việc và kết quả thu được để đưa ra báo cáo kết quả, đề xuất các phương án, giải pháp để khắc phục các lỗ hổng, điểm yếu bảo mật.

2.3.1. Thành phần trong tập luật của MobSF

Để tích hợp quy trình phát hiện lỗ hổng mã nguồn, cách tiếp cận theo tập luật từ công cụ Mobsf nhằm giải quyết nguy cơ về ATTT. Dưới đây là một số các luật liên quan đến lỗ hổng mã nguồn và phân loại theo các quy tắc thường được áp dụng để phát hiện lỗ hổng mã nguồn :

1. Quy tắc về bảo mật:

Write app directory là ứng dụng có quyền ghi vào thư mục của chính nó và trong đó có thông báo rằng thông tin nhạy cảm nên được mã hóa.

- Quyền ghi vào thư mục ứng dụng: Ứng dụng có quyền ghi vào thư mục của chính mình, điều này thường xuyên là trường hợp để lưu trữ dữ liệu cần thiết cho ứng dụng.
- Mã hóa thông tin nhạy cảm: Do thư mục của ứng dụng có thể bị truy cập bởi các ứng dụng khác hoặc người dùng có quyền truy cập vào thiết bị, thông tin nhạy cảm nên được mã hóa để bảo vệ nó khỏi truy cập trái phép.
- Quản lý khóa và mật khẩu: Để giải mã thông tin được mã hóa, ứng dụng cần sử dụng một hệ thống quản lý khóa và mật khẩu an toàn. Việc này đảm bảo rằng chỉ những người có quyền truy cập mới có thể giải mã thông tin.
- Chăm sóc an toàn: Đảm bảo rằng quá trình mã hóa và giải mã được thực hiện theo cách an toàn. Sử dụng các thư viện mã hóa được chứng minh và thực hiện các biện pháp bảo mật như bảo vệ khóa và tránh các lỗ hổng bảo mật có thể xảy ra trong mã nguồn.
- Bảo mật mạng: Nếu thông tin nhạy cảm được truyền qua mạng, đảm bảo rằng kết nối là an toàn bằng cách sử dụng giao thức HTTPS và bảo vệ dữ liệu trong quá trình truyền tải.
- Tuân thủ quy định bảo mật: Tuân thủ các quy định và chuẩn bảo mật đặc thù cho nền tảng hoặc ngành công nghiệp cụ thể nơi ứng dụng được triển khai.

Những biện pháp an toàn như trên giúp đảm bảo rằng dữ liệu nhạy cảm được bảo vệ và chỉ có những người có quyền truy cập mới có thể truy xuất được thông tin này.

SQL cipher là ứng dụng sử dụng SQL Cipher (một thư viện mã hóa cho cơ sở dữ liệu SQLite), và đồng thời cảnh báo về việc không nên lưu trữ các thông tin bí mật trực tiếp trong mã nguồn của ứng dụng.

Dưới đây là một số điểm quan trọng:

- SQL Cipher: là một thư viện mã hóa cơ sở dữ liệu SQLite, giúp bảo vệ dữ liệu trong cơ sở dữ liệu bằng cách sử dụng các thuật toán mã hóa mạnh mẽ.

- Thông tin bí mật: như các khóa API, mật khẩu, hoặc thông tin xác thực, không nên được đặt trực tiếp trong mã nguồn của ứng dụng. Việc này làm tăng rủi ro bảo mật vì nếu mã nguồn bị lộ, thông tin bí mật cũng bị rủi ro.
- Sử dụng các cơ chế an toàn: nên sử dụng các cơ chế an toàn như bộ quản lý bí mật (Key Vault), biến môi trường, hoặc các phương tiện khác để lưu trữ và quản lý.
- Tuân Thủ Chuẩn Bảo Mật: Đảm bảo rằng ứng dụng tuân thủ các chuẩn bảo mật và quy tắc an toàn liên quan đến việc xử lý thông tin nhạy cảm và secrets.
- Quản Lý Khóa và Mật Khẩu An Toàn: Khi sử dụng SQL Cipher hoặc bất kỳ thư viện mã hóa nào khác, đảm bảo rằng quản lý khóa và mật khẩu cũng được thực hiện một cách an toàn.

Những biện pháp trên giúp đảm bảo rằng dữ liệu trong cơ sở dữ liệu được bảo vệ bằng SQL Cipher, và thông tin bí mật không bị rò rỉ trong mã nguồn.

Aar jar debug enabled cảnh báo về việc cấu hình gỡ lỗi đã được kích hoạt trong ứng dụng. Nó nêu rõ rằng trong bản build dành cho môi trường sản xuất, không nên cho phép khả năng gỡ lỗi.

Dưới đây là một số điểm quan trọng liên quan đến cảnh báo này:

- Rủi Ro Bảo Mật: Bản sản xuất nên được thiết lập để không thể gỡ lỗi, vì việc kích hoạt tính năng gỡ lỗi có thể tạo ra rủi ro bảo mật. Kẻ tấn công có thể sử dụng thông tin gỡ lỗi để phân tích mã nguồn và tìm ra các lỗ hổng bảo mật.
- Xây dựng cấu hình: Cấu hình cần được điều chỉnh một cách chặt chẽ để đảm bảo rằng tính năng gỡ lỗi không được kích hoạt trong bản build cuối cùng dành cho người dùng.
- Kiểm tra cấu hình: Trước khi triển khai ứng dụng, nhà phát triển cần kiểm tra cấu hình build để đảm bảo rằng khả năng gỡ lỗi đã được tắt trong bản sản xuất.
- Tuân thủ chuẩn bảo mật: Tuân thủ các chuẩn bảo mật và các hướng dẫn của nền tảng để đảm bảo rằng ứng dụng không có khả năng gỡ lỗi trong môi trường sản xuất.

Những biện pháp trên giúp đảm bảo tính an toàn và bảo mật cho ứng dụng khi triển khai trong môi trường sản xuất.

Debugger detect phát hiện mã nguồn trong ứng dụng đã được bảo vệ bằng DexGuard, và mã nguồn này chứa mã để phát hiện sự tồn tại của bộ gỡ lỗi trong quá trình thực thi ứng dụng.

DexGuard là một công cụ bảo vệ mã nguồn (obfuscation and code hardening) được sử dụng chủ yếu trong môi trường Android để bảo vệ ứng dụng khỏi việc phân tích ngược và tấn công. Một trong những tính năng quan trọng của DexGuard là khả năng nhận diện và ngăn chặn việc sử dụng bộ gỡ lỗi trong quá trình chạy ứng dụng[5, tr.2180].

Khi thông báo "DexGuard Debugger Detection code is identified." xuất hiện, điều này có nghĩa là trong quá trình kiểm tra mã nguồn của ứng dụng bảo vệ bởi DexGuard, đã phát hiện mã nguồn chứa các phần mã được tích hợp bởi DexGuard để phát hiện sự hiện diện của bộ gỡ lỗi. Điều này thường được thực hiện để ngăn chặn việc phân tích và gỡ lỗi ứng dụng, làm tăng độ khó cho kẻ tấn công muốn hiểu và thay đổi mã nguồn của ứng dụng.

Tính năng này là một phần của chiến lược bảo vệ ứng dụng để đảm bảo tính toàn vẹn và bảo mật của mã nguồn khi chạy trong môi trường Android.

Emulator detect phát hiện mã nguồn trong ứng dụng đã được bảo vệ bằng DexGuard, và mã nguồn này chứa các đoạn mã được thiết kế để phát hiện sự chạy trên máy ảo.

DexGuard là một công cụ bảo vệ mã nguồn thường được sử dụng trong môi trường Android để bảo vệ ứng dụng khỏi việc phân tích ngược và tấn công. Trong bối cảnh này, nó cũng có khả năng nhận diện việc ứng dụng đang chạy trên máy ảo, chẳng hạn như Android Emulator, và có thể thực hiện các biện pháp bảo vệ phù hợp[2, tr.50].

Khi thông báo "DexGuard Emulator Detection code is identified." xuất hiện, đó có nghĩa là trong quá trình kiểm tra mã nguồn của ứng dụng đã được bảo vệ bởi

DexGuard, đã phát hiện mã nguồn chứa các phần mã được tích hợp để xác định xem ứng dụng có đang chạy trên máy ảo hay không. Điều này có thể được thực hiện để ngăn chặn việc thử nghiệm và phân tích ứng dụng trên môi trường giả lập.

Tính năng này giúp bảo vệ ứng dụng khỏi các kỹ thuật tấn công mà kẻ tấn công có thể sử dụng khi chạy trên các môi trường máy ảo, giúp tăng cường bảo mật của ứng dụng trong môi trường Android.

Debug sign khi phát triển và kiểm thử ứng dụng Android, thường sử dụng một chìa khóa gỡ lỗi để ký ứng dụng, giúp việc gỡ lỗi và theo dõi thông tin trên thiết bị Android[6, tr.10].

Các bước để xác định xem ứng dụng có được ký bằng chìa khóa gỡ lỗi hay không thường bao gồm việc kiểm tra chữ ký của ứng dụng và so sánh với chữ ký của chìa khóa gỡ lỗi mặc định.

Một số biện pháp phổ biến có thể được thực hiện trong mã nguồn của ứng dụng hoặc trong tệp build.gradle để xác định xem ứng dụng có được ký bằng chìa khóa gỡ lỗi hay không.

Điều này giúp nhận biết xem ứng dụng có đang chạy trong chế độ gỡ lỗi hay không và thực hiện các hành động phù hợp, như thông báo hoặc việc kiểm soát quyền truy cập. Đoạn mã này có thể được bảo vệ bằng DexGuard để ngăn chặn việc tấn công từ những kẻ tấn công có thể muốn thay đổi hành vi của ứng dụng khi chạy trong chế độ gỡ lỗi.

Dexguard root detection cho biết mã nguồn liên quan đến việc phát hiện thiết bị đã được root trong ứng dụng Android được bảo vệ bằng DexGuard đã được phát hiện. Trong ngữ cảnh này:

- Root Detection: Là mã nguồn hoặc phần của ứng dụng được thiết kế để xác định xem thiết bị đang chạy có được root (quyền quản trị hệ thống) hay không. Việc kiểm tra root thường được thực hiện để tăng cường bảo mật hoặc ngăn chặn việc sử dụng không ủy quyền của ứng dụng.

- Nếu có thông báo "is identified", nghĩa là có sự nhận diện, phát hiện ra rằng mã nguồn liên quan đến việc kiểm tra root đã được xác định hoặc được phát hiện.

Trong bối cảnh bảo mật, việc phát hiện mã nguồn kiểm tra root có thể là một vấn đề, vì những ứng dụng thường muốn tránh chạy trên các thiết bị đã root để ngăn chặn việc thay đổi không ủy quyền có thể đe dọa tính ổn định và bảo mật của ứng dụng.

2. Quy tắc về Hiệu suất:

Tamper detect mã liên quan đến việc phát hiện sự thay đổi trong ứng dụng được bảo vệ bằng DexGuard đã được xác định hoặc phát hiện ra. Trong ngữ cảnh này:

- DexGuard: Là một giải pháp bảo mật cho ứng dụng Android được phát triển bởi Guardsquare. DexGuard cung cấp các tính năng bảo mật như mã hóa mã nguồn, làm mờ mã nguồn, và phát hiện sự thay đổi để bảo vệ ứng dụng khỏi các mối đe dọa bảo mật[7, tr.1020].
- App Tamper Detection code: Là mã nguồn hoặc một phần của ứng dụng được thiết kế để phát hiện sự thay đổi không ủy quyền trong mã nguồn hoặc dữ liệu của ứng dụng. Tampering detection thường được sử dụng để ngăn chặn những thay đổi không đáng kể có thể ảnh hưởng đến tính toàn vẹn và bảo mật của ứng dụng[1, tr.2].
- Nếu có thông báo "is identified", nghĩa là có sự nhận diện, phát hiện ra rằng mã nguồn liên quan đến việc phát hiện sự thay đổi trong ứng dụng đã được xác định hoặc được phát hiện.

Trong bảo mật, việc phát hiện tampering giúp đảm bảo rằng ứng dụng không bị sửa đổi một cách không ủy quyền, giữ cho nó an toàn và đáng tin cậy trước các mối đe dọa từ việc thay đổi mã nguồn hoặc tài nguyên của ứng dụng.

3. Quy tắc về Độ tin cậy:

WebView thực thi mã do người dùng kiểm soát trong WebView là một lỗ

hổng bảo mật nghiêm trọng. Cảnh báo về một vấn đề bảo mật trong cách ứng dụng xử lý và sử dụng WebView. Dưới đây là một số điểm quan trọng:

- WebView Implementation: WebView là một thành phần trong phát triển ứng dụng di động, thường được sử dụng để hiển thị nội dung web trong ứng dụng. Triển khai WebView không an toàn có thể mở cửa cho các lỗ hổng bảo mật.
- Execution of User-Controlled Code: Lỗ hổng bảo mật được đề cập đến ở đây là khả năng thực thi mã do người dùng kiểm soát trong WebView. Nếu ứng dụng cho phép thực hiện mã JavaScript hoặc nội dung web có chứa mã nguyên trực tiếp từ người dùng, có nguy cơ lớn về tấn công XSS (Cross-Site Scripting) và các vấn đề liên quan đến việc thực thi mã không an toàn[4, tr16].
- Cross-Site Scripting (XSS): XSS là một loại tấn công bảo mật mà kẻ tấn công chèn mã độc hại vào trang web hoặc ứng dụng, và sau đó thực hiện nó trên thiết bị của người dùng khi trang web hoặc ứng dụng được tải[3, tr.1].
- Phương tiện kiểm soát mã: Đối với WebView, quản lý và kiểm soát mã nguồn là rất quan trọng. Tất cả các đầu vào từ người dùng cần được kiểm tra và xử lý một cách an toàn để ngăn chặn việc chèn mã độc hại.
- Content Security Policy: Sử dụng các chính sách an toàn như Content Security Policy để hạn chế việc thực thi mã JavaScript và các nguồn tài nguyên từ các nguồn không an toàn.
- WebView Hardening: Tối ưu hóa cấu hình WebView và thực hiện các biện pháp tăng cường an ninh để giảm thiểu rủi ro tấn công thông qua WebView.

Một triển khai WebView không an toàn có thể tạo ra nhiều rủi ro an ninh, đặc biệt là khi thực thi mã do người dùng kiểm soát. Để bảo vệ ứng dụng, quản lý mã nguồn và triển khai các biện pháp bảo mật là rất quan trọng.

4. Quy tắc về Kiểm soát biên:

Logging là cảnh báo về việc ứng dụng đang ghi lại thông tin trong quá trình chạy, nhưng thông tin nhạy cảm không nên xuất hiện trong nhật ký. Điều này là quan trọng để bảo vệ thông tin cá nhân và giữ cho ứng dụng tuân thủ các chuẩn bảo mật và quy định về quyền riêng tư.

Dưới đây là một số lý do và biện pháp cần được thực hiện:

- Bảo mật thông tin nhạy cảm: Thông tin nhạy cảm như mật khẩu, thông tin tài khoản người dùng, hoặc thông tin cá nhân không nên xuất hiện trong các bản ghi nhật ký. Việc lưu trữ thông tin này có thể tạo ra rủi ro bảo mật nếu nhật ký bị lạc hoặc bị truy cập trái phép.
- Chống tấn công kiểm soát nhật ký: Việc ghi thông tin nhạy cảm có thể là một điểm đầu tiên mà kẻ tấn công có thể tận dụng để thu thập thông tin đối tượng. Điều này có thể đặt người dùng và hệ thống trong nguy cơ.
- Tuân thủ quy định bảo mật và quyền riêng tư: Nhiều quy định và chuẩn bảo mật yêu cầu việc bảo vệ thông tin nhạy cảm và đảm bảo rằng nó không bị ghi vào nhật ký một cách dễ dàng.
- Kiểm soát cấp độ nhật ký: Đối với các thông tin không nhạy cảm, việc ghi vào nhật ký có thể giúp theo dõi lỗi và cải thiện quản lý hệ thống. Tuy nhiên, cần kiểm soát cấp độ nhật ký để đảm bảo rằng thông tin nhạy cảm không được ghi.
- Kiểm tra mã nguồn: Kiểm tra mã nguồn của ứng dụng để đảm bảo rằng không có thông tin nhạy cảm nào được ghi vào nhật ký một cách không an toàn.

Bằng cách thực hiện những biện pháp trên, ứng dụng có thể giữ cho nhật ký an toàn và đảm bảo rằng thông tin nhạy cảm không bao giờ được ghi vào đó.

5. Quy tắc về Tiêu Chuẩn Mã:

Insecure random là ứng dụng sử dụng một công cụ tạo số ngẫu nhiên không an toàn. Một công cụ tạo số ngẫu nhiên không an toàn có thể tạo ra các số ngẫu nhiên không đều, dự đoán được, hoặc có thể bị dự đoán bởi người khác. Trong lập trình và bảo mật thông tin, việc sử dụng một hàm tạo số ngẫu nhiên không an toàn có thể tạo ra nhiều vấn đề bảo mật.

Một số vấn đề có thể xuất hiện khi ứng dụng sử dụng một công cụ tạo số ngẫu nhiên không an toàn bao gồm:

- Dự đoán được: Nếu thuật toán tạo số ngẫu nhiên không an toàn, thì kết quả có thể dự đoán được. Điều này có thể làm giảm hiệu suất bảo mật của ứng dụng, đặc biệt là trong các trường hợp như sinh khóa mã hóa.
- Khả năng dự đoán và tấn công: Nếu kẻ tấn công biết cách dự đoán các giá trị ngẫu nhiên, họ có thể tận dụng điều này để thực hiện các cuộc tấn công, chẳng hạn như tấn công theo dõi hoặc tấn công kiểm soát.
- Bảo mật hệ thống: Số ngẫu nhiên thường được sử dụng trong nhiều ngữ cảnh để tăng cường bảo mật, chẳng hạn như việc tạo khóa mã hóa. Nếu số ngẫu nhiên không an toàn, có thể ảnh hưởng đến mức độ bảo mật của hệ thống.

Để giải quyết vấn đề này, lập trình viên thường nên sử dụng các thư viện và hàm tạo số ngẫu nhiên được chứng minh và được coi là an toàn. Trong môi trường lập trình Java, ví dụ, có thể sử dụng java.security.SecureRandom thay vì java.util.Random để tạo số ngẫu nhiên an toàn. Việc sử dụng các nguồn ngẫu nhiên an toàn giúp đảm bảo tính dự đoán khó khăn và cải thiện bảo mật của ứng dụng.

6. Quy tắc về Ngôn Ngữ và API:

Safetynet cho biết ứng dụng này sử dụng SafetyNet API. Dưới đây là một số giải thích liên quan:

- SafetyNet API: Đây là một API (Application Programming Interface) được cung cấp bởi Google cho các nhà phát triển ứng dụng Android. API này giúp ứng dụng xác định tính toàn vẹn của môi trường thiết bị Android mà nó đang chạy.
- Tính toàn vẹn thiết bị: SafetyNet API có thể được sử dụng để kiểm tra xem thiết bị Android có đang chạy trong môi trường an toàn và không bị root (đã được cài đặt quyền root) hay không. Nó cũng cung cấp thông tin về sự thay đổi trong hệ điều hành hoặc ứng dụng đã được thực hiện, giúp nhận diện các thiết bị bị tấn công hoặc đã được sửa đổi một cách không ủy quyền.
- Sử dụng trong ứng dụng: Khi một ứng dụng sử dụng SafetyNet API, nó thường được áp dụng để tăng cường bảo mật và chống gian lận. Ví dụ, một ứng dụng

ngân hàng có thể sử dụng SafetyNet API để đảm bảo rằng nó đang chạy trên một thiết bị không bị root và không bị tấn công.

Tổng quát, việc sử dụng SafetyNet API giúp các nhà phát triển ứng dụng đảm bảo rằng ứng dụng của họ đang chạy trong một môi trường an toàn và ngăn chặn những hành vi gian lận hoặc đánh cắp thông tin.

7. Quy tắc về Giao Diện Người Dùng (UI):

Hidden UI là các phần tử ẩn trong giao diện người dùng có thể được sử dụng để che giấu thông tin khỏi người dùng, nhưng rủi ro là thông tin này có thể bị rò rỉ hoặc bị tiết lộ ra bên ngoài. Trong lập trình web hoặc phát triển ứng dụng, các phần tử ẩn thường được sử dụng để lưu trữ dữ liệu hoặc giữ các thành phần không mong muốn nằm ngoài tầm nhìn của người dùng. Tuy nhiên, nếu không có các biện pháp bảo mật phù hợp, thông tin này có thể trở nên khả dụng cho kẻ tấn công hoặc người dùng cuối có kiến thức kỹ thuật cao.

Một số nguy cơ và phương pháp rò rỉ thông tin có thể bao gồm:

- Inspect Element: Người dùng có thể sử dụng chức năng "Inspect" trong trình duyệt để kiểm tra mã nguồn trang web và tìm thấy thông tin được ẩn.
- Network Requests: Thông tin có thể được rò rỉ qua các yêu cầu mạng, và người dùng có thể kiểm tra các yêu cầu này để xem thông tin bổ sung.
- CSS Manipulation: Bằng cách sử dụng kỹ thuật CSS, người dùng có thể thay đổi các thuộc tính của phần tử ẩn để làm cho nó hiển thị và tiết lộ thông tin.
- Browser Extensions: Các tiện ích mở rộng trình duyệt có thể được sử dụng để thay đổi hoặc hiển thị thông tin ẩn.
- Cross-Site Scripting (XSS): Nếu ứng dụng web không an toàn trước tấn công XSS, người tấn công có thể chèn mã độc hại để truy cập hoặc thay đổi thông tin ẩn.

Để ngăn chặn rủi ro này, nhà phát triển cần thực hiện biện pháp bảo mật như mã hóa thông tin, kiểm tra chính xác dữ liệu trước khi hiển thị, và sử dụng các kỹ thuật an ninh mạng để bảo vệ dữ liệu truyền qua mạng. Hơn nữa, việc kiểm tra và xử

lý các lỗ hổng bảo mật trong mã nguồn là quan trọng để đảm bảo an toàn cho thông tin của người dùng.

2.3.2. Phương pháp kiểm tra mã nguồn của tập luật trong MobSF

Nguyên lý hoạt động của phương pháp kiểm tra mã nguồn của tập luật trong công cụ rà quét tự động thường bao gồm các bước chính sau:

1. Thu thập mã nguồn: Công cụ sẽ thu thập mã nguồn của ứng dụng hoặc dự án từ nguồn cung cấp (ví dụ: tệp mã nguồn, repository Git). Đối với ứng dụng di động, có thể là tệp tin APK (Android) hoặc IPA (iOS).

2. Phân đoạn (parsing) và phân tích cú pháp: Công cụ sẽ phân tích cú pháp của mã nguồn để hiểu cấu trúc và ngữ pháp của mã. Điều này thường bao gồm việc sử dụng lexer và parser để chuyển đổi mã nguồn thành cây cú pháp.

3. Xây dựng đồ thị ưu tiên (Abstract Syntax Tree - AST): Công cụ tạo ra một đồ thị ưu tiên của mã nguồn, thường được biết đến là Abstract Syntax Tree (AST). AST biểu diễn cấu trúc cú pháp của mã nguồn và là cơ sở cho việc phân tích tiếp theo.

4. Kiểm tra lỗ hổng tĩnh (Static Analysis): Công cụ sử dụng các kỹ thuật phân tích tĩnh để kiểm tra mã nguồn mà không cần thực hiện thực thi chương trình. Các kỹ thuật này bao gồm:

- Kiểm tra quy tắc lập trình (Code Rule Checking): Kiểm tra xem mã nguồn có tuân thủ các quy tắc lập trình an toàn hay không.
- Phân tích luồng dữ liệu (Data Flow Analysis): Theo dõi cách dữ liệu được truyền đi trong mã nguồn để xác định các lỗ hổng liên quan đến dữ liệu nhạy cảm.
- Phân tích luồng điều khiển (Control Flow Analysis): Phân tích cách chương trình chuyển đổi giữa các câu lệnh để xác định các vấn đề an toàn điều khiển.

5. Phát hiện lỗ hổng bảo mật: Công cụ sử dụng cơ sở dữ liệu chứa các luật và quy tắc lỗ hổng bảo mật để so sánh với mã nguồn và phát hiện các mô hình hoặc cấu trúc mã nguồn có thể liên quan đến các lỗ hổng bảo mật.

6. Tạo báo cáo và thông báo: Sau khi hoàn tất quá trình phân tích, công cụ tạo ra báo cáo chi tiết về các lỗ hổng bảo mật phát hiện được. Báo cáo này thường bao gồm mô tả vấn đề, địa chỉ của nó trong mã nguồn, và đôi khi cung cấp giải pháp sửa chữa.

Hiệu suất và độ chính xác của phân tích tĩnh phụ thuộc vào độ phức tạp của mã nguồn, khả năng cấu hình của công cụ, và cập nhật thường xuyên của cơ sở dữ liệu lỗ hổng bảo mật. Các công cụ phân tích tĩnh mã nguồn như SonarQube, Checkmarx, hoặc Fortify thường áp dụng các nguyên tắc và kỹ thuật tương tự để phát hiện lỗ hổng bảo mật trong mã nguồn.

Trong công cụ MobSF phần quản lý tập luật được chia thành bốn phần do các tệp khác nhau quản lý bao gồm: tệp android_apis.yaml, android_niap.yaml, android_permissions.yaml và android_rules.yaml. Các tệp này có mối quan hệ trong quá trình triển khai, rà quét , đối chiếu các lỗ hổng phát hiện được theo các luật trong tập luật.

Android_apis.yaml là tệp chứa thông tin về các API (Application Programming Interface) của hệ điều hành Android. Các API cung cấp các giao diện để ứng dụng có thể tương tác với hệ thống và các dịch vụ của Android. Trong quá trình phát triển ứng dụng, các nhà phát triển có thể sử dụng tệp android_apis.yaml để tìm hiểu về các API có sẵn và cách sử dụng chúng trong ứng dụng.

Android_niap.yaml là tệp này chứa thông tin về việc tuân thủ các tiêu chuẩn bảo mật do National Information Assurance Partnership (NIAP) quản lý cho hệ điều hành Android. Trong quá trình phát triển ứng dụng, các nhà phát triển có thể sử dụng tệp android_niap.yaml để xác định các yêu cầu bảo mật và tiêu chuẩn cần tuân thủ để đạt được chứng nhận bảo mật NIAP.

Android_permissions.yaml là tệp chứa thông tin về các quyền mà ứng dụng yêu cầu để truy cập vào các tài nguyên hoặc chức năng cụ thể của thiết bị Android. Trong quá trình phát triển ứng dụng, các nhà phát triển có thể sử dụng tệp android_permissions.yaml để quản lý và tài liệu hóa các quyền truy cập của ứng dụng.

Android_rules.yaml là tệp chứa các quy tắc hoặc các điều luật được áp dụng trong quá trình phát triển, kiểm tra hoặc triển khai ứng dụng trên nền tảng Android. Trong quá trình phát triển ứng dụng, các nhà phát triển có thể sử dụng tệp android_rules.yaml để xác định các quy tắc và điều luật cần tuân thủ và áp dụng chúng trong quá trình phát triển và kiểm tra ứng dụng.

2.4. Kết luận chương 2

Phát hiện lỗ hổng mã nguồn là một bước quan trọng để ngăn chặn, giảm thiểu các rủi ro gây hại cho ứng dụng và khả năng bị lộ lọt thông tin hay mất dữ liệu. Cách phát hiện lỗ hổng là: sử dụng công cụ và phân tích báo cáo rà quét. Chương tiếp theo sẽ trình bày chi tiết quá trình phát hiện và đánh giá lỗ hổng mã nguồn cho một ứng dụng bao gồm các bước cài đặt, cấu hình công cụ, tạo kịch bản thử nghiệm, phân tích báo cáo và đánh giá kết quả.

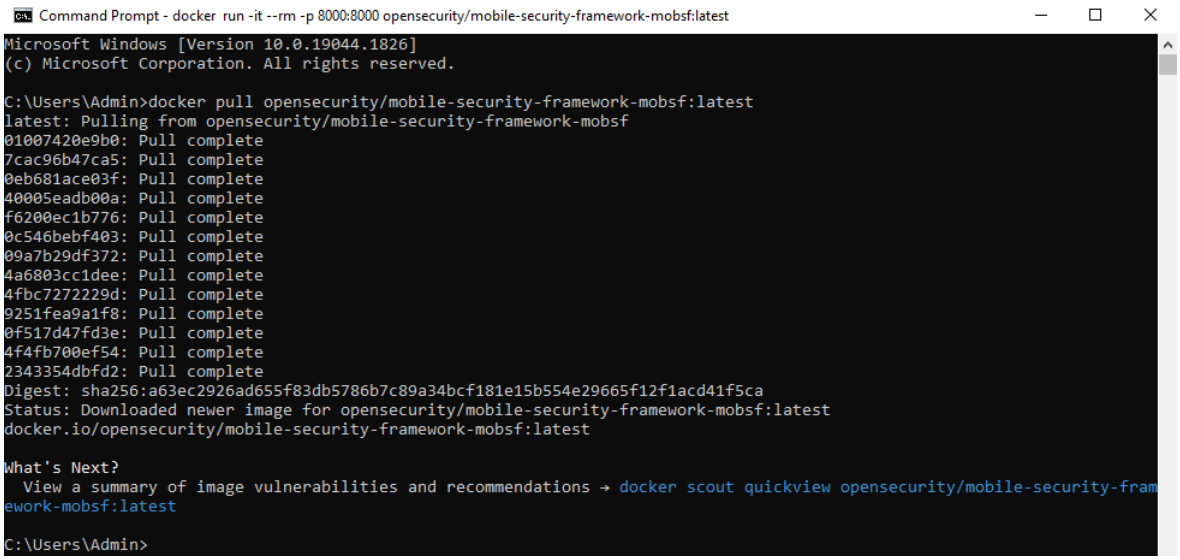
CHƯƠNG 3: THỰC NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

3.1. Cài đặt và cấu hình công cụ

Để cài đặt công cụ MobSF, cần phải cài đặt docker để chạy được MobSF và làm theo lệnh:

docker pull opensecurity/mobile-security-framework-mobsf:latest

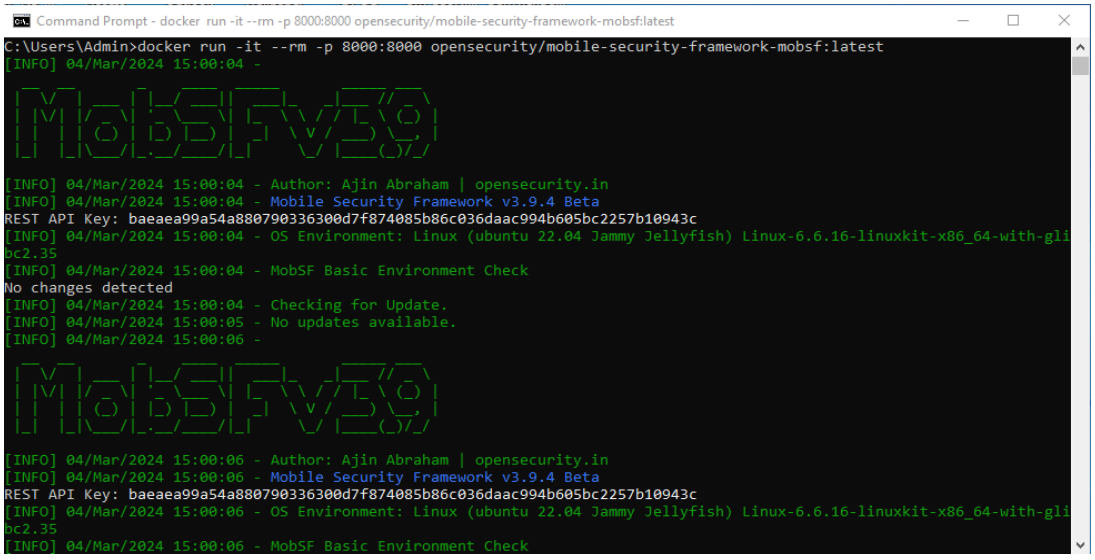
Cần tải và cài đặt phiên bản cuối của MobSF để có phiên bản, tính năng, bảo mật và sửa lỗi cập nhật nhất



Hình 3.1. Tải và cài đặt công cụ MobSF

Tiến hành triển khai công cụ MobSF với hỗ trợ phân tích tĩnh sử dụng câu lệnh sau:

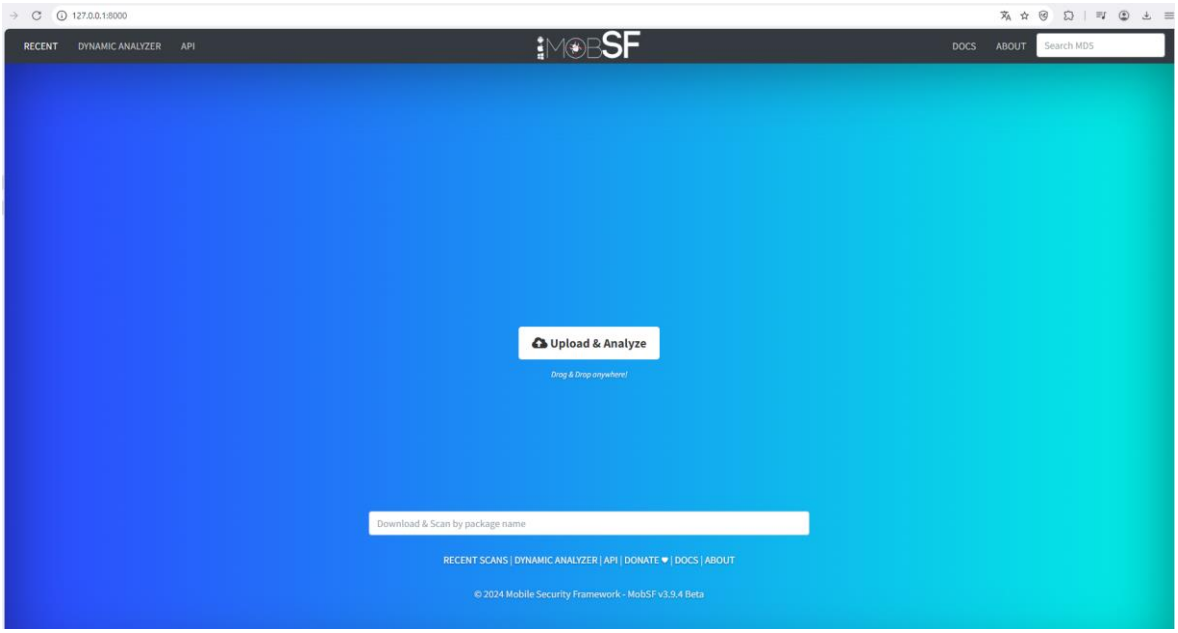
docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest



Hình 3.2. Triển khai công cụ MobSF trên máy chủ cục bộ

Bây giờ, để chạy MobSF thực hiện truy cập MobSF http://127.0.0.1:8000 trong trình duyệt web và MobSF sẽ chạy trên một máy chủ cục bộ trên cổng 8000.

Mở liên kết trong trình duyệt và xem MobSF đã được cài đặt



Hình 3.3. Hoàn thành cài đặt công cụ MobSF

3.2. Kịch bản thử nghiệm

Từ quy trình phát hiện lỗ hổng được trình bày ở trên, kịch bản thử nghiệm cho việc rà quét, đánh giá lỗ hổng bảo mật cho một phần mềm, ứng dụng cụ thể được lập và hoàn thành như sau:

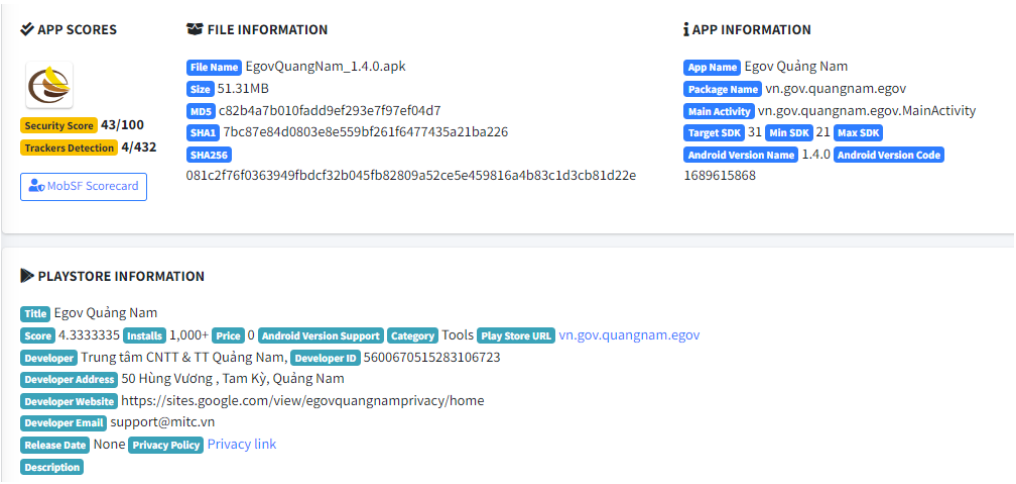
Bảng 3.1. Kịch bản thử nghiệm phát hiện lỗ hổng mã nguồn

Tên nhiệm vụ	Mô tả nhiệm vụ	Kết quả mong đợi
Bước 1: Chuẩn bị đánh giá		
Nhiệm vụ 1-1: Xác định mục đích	Tìm kiếm các lỗ hổng mã nguồn từ đó khắc phục các lỗ hổng và cải thiện độ tin cậy của phần mềm Egov Quảng Nam.	Đơn vị phát triển cung cấp tệp cài đặt đóng gói của phần mềm Egov Quảng Nam phiên bản Android.
Nhiệm vụ 1-2: Xác định phạm vi	Phát hiện lỗ hổng mã nguồn của phần mềm Egov Quảng Nam phiên bản Android, trong quá trình rà quét sẽ sử dụng công cụ MobSF.	

Nhiệm vụ 1-3: Xác định các đánh giá và hạn chế	Phát hiện các lỗ hổng trong mã nguồn của phần mềm Egov Quảng Nam phiên bản Android và yêu cầu bản vá đã cập nhật khắc phục lỗ hổng có mức độ nguy hại cao.	
Nhiệm vụ 1-4: Xác định nguồn thông tin	Phân tích chi tiết trong Báo cáo rà quét lỗ hổng bảo mật tại mục 3.3.	
Bước 2: Thực hiện đánh giá		
Nhiệm vụ 2-1: Xác định nguồn gốc	Phân tích chi tiết trong Báo cáo rà quét lỗ hổng bảo mật tại mục 3.3.	Rà quét, phân tích và xuất báo cáo để đơn vị phát triển khắc phục.
Nhiệm vụ 2-2: Xác định sự kiện	Phân tích chi tiết trong Báo cáo rà quét lỗ hổng bảo mật tại mục 3.3.	
Nhiệm vụ 2-3: Xác định nhiệm vụ và điều kiện bảo đảm	Các lỗ hổng có cảnh báo mức độ nguy hại cao gây ảnh hưởng đến an toàn bảo mật sẽ được yêu cầu khắc phục. Các cảnh báo còn lại sẽ khắc phục sau trong quá trình vận hành thực tế.	
Nhiệm vụ 2-4: Xác định các tác động	Phân tích chi tiết trong Báo cáo rà quét lỗ hổng bảo mật tại mục 3.3.	
Nhiệm vụ 2-5: Xác định rủi ro	Phân tích chi tiết trong Báo cáo rà quét lỗ hổng bảo mật tại mục 3.3.	
Bước 3: Truyền đạt kết quả		
Nhiệm vụ 3-1: Truyền đạt kết quả	Gửi báo cáo rà quét và bản phân tích để đơn vị phát triển khắc phục.	Đơn vị phát triển tiếp nhận báo cáo và khắc phục lỗ hổng trong phần mềm.
Nhiệm vụ 3-2: Chia sẻ thông tin liên quan	Đơn vị phát triển khắc phục và xử lý các lỗ hổng có cảnh báo mức độ nguy hại cao.	
Bước 4: Duy trì		
Nhiệm vụ 4-1: Giám sát các yếu tố rủi ro	Đơn vị phát triển gửi lại bản cập nhật để tiến hành rà quét và đánh giá lại phần mềm.	Rà quét, đánh giá và cập nhật lại báo cáo kết quả.
Nhiệm vụ 4-2: Cập nhật đánh giá	Cập nhật đánh giá rủi ro hiện có.	

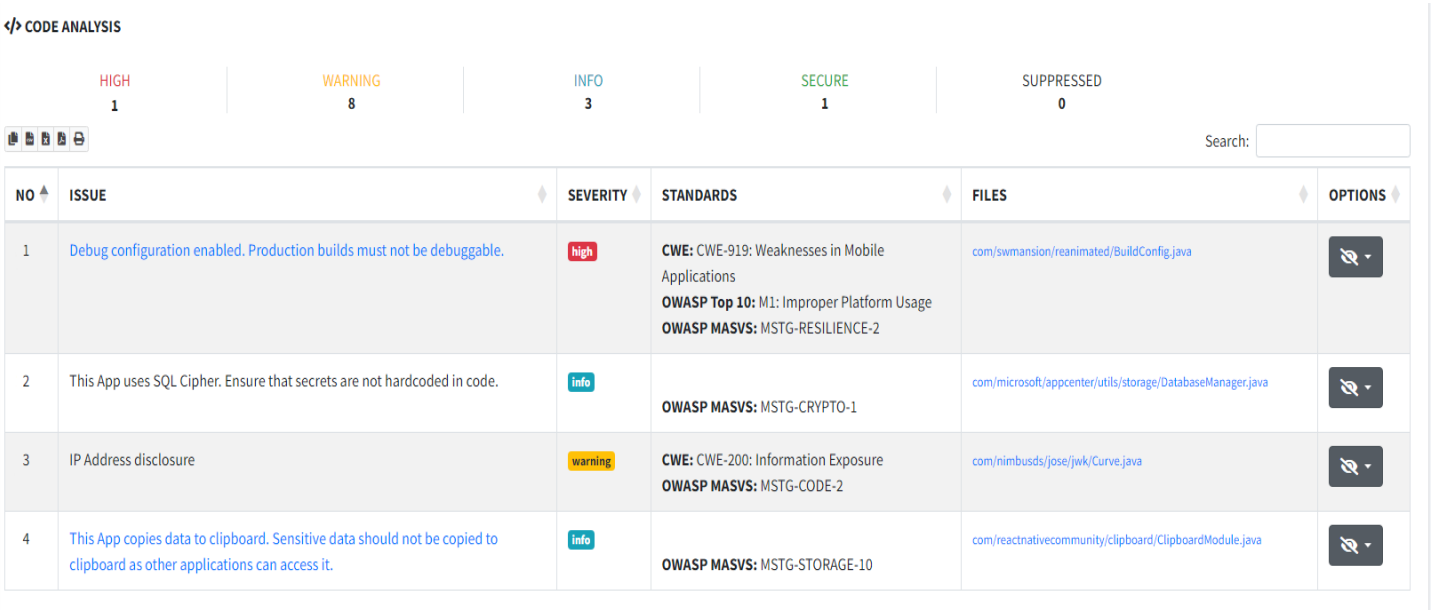
3.3. Thực nghiệm và đánh giá

Dựa trên kịch bản thử nghiệm để tiến hành rà quét phát hiện lỗ hổng mã nguồn của phần mềm Egov Quảng Nam phiên bản Android. Đầu tiên, tệp apk của phần mềm sẽ được rà quét bằng công cụ MobSF.



Hình 3.4. Thông tin phần mềm phiên bản 1.4.0

Sau khi rà quét, báo cáo phân tích mã nguồn chi tiết được xuất ra như sau:



Hình 3.5. Báo cáo phân tích mã nguồn của phần mềm phiên bản 1.4.0

Trong báo cáo phân tích chi tiết phần mềm, công cụ đã phân tích, biên dịch

ngược mã máy từ file APK thành các file .java để rà quét lỗ hổng mã nguồn theo tập luật trong từng file.

BuildConfig.java

```java
package com.swmansion.reanimated;
/* loaded from: classes3.dex */
public final class BuildConfig {
    public static final String BUILD_TYPE = "debug";
    public static final boolean DEBUG = true;
    public static final int EXOPACKAGE_FLAGS = 0;
    public static final boolean IS_INTERNAL_BUILD = false;
    public static final boolean IS_NEW_ARCHITECTURE_ENABLED = false;
    public static final String LIBRARY_PACKAGE_NAME = "com.swmansion.reanimated";
    public static final int REACT_NATIVE_MINOR_VERSION = 68;
}
```

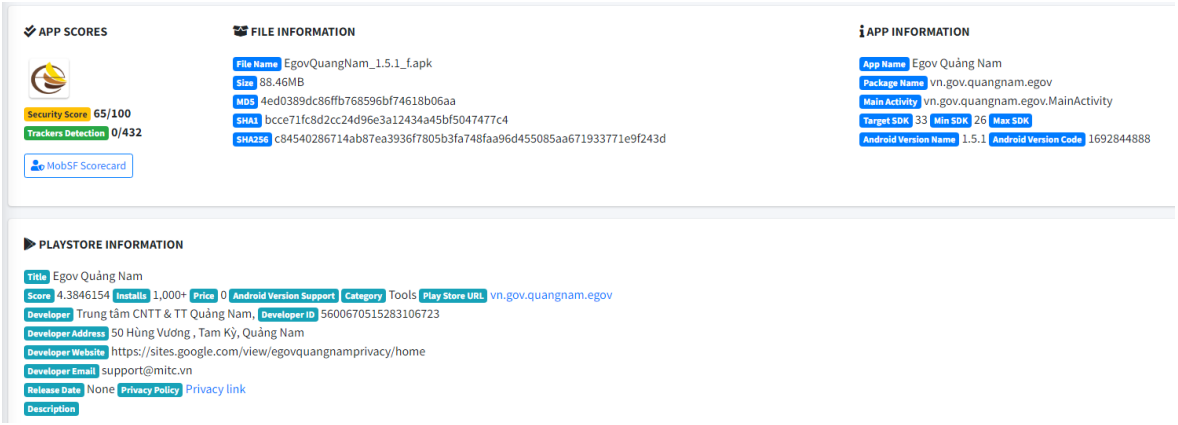
Hình 3.6. Tệp mã nguồn phát hiện lỗ hổng hệ thống

Theo kết quả trên, các lỗ hổng mã nguồn có mức độ nguy hại cao phát hiện được trong phiên bản 1.4.0 của phần mềm Egov Quảng Nam có 1 lỗ hổng:

1. android_aar_jar_debug_enabled
   - Mức độ cảnh báo: Cao
   - Tiêu chuẩn:
     - CWE: CWE-919: Weaknesses in Mobile Applications
     - OWASP Top 10: M1: Improper Platform Usage
     - OWASP MASVS: MSTG-RESILIENCE-2
   - Đường dẫn: com/swmansion/reanimated/BuildConfig.java
   - Mô tả: Chế độ gỡ lỗi thường cung cấp thông tin chi tiết về mã nguồn và hoạt động của ứng dụng, điều này có thể tạo ra lỗ hổng bảo mật và có thể được lợi dụng bởi những người có ý định xấu.
   - Tác động: Việc đặt cờ android:debuggable thành true sẽ cho phép kẻ tấn công gỡ lỗi ứng dụng, giúp chúng dễ dàng truy cập vào các phần của ứng dụng cần được bảo mật.
   - Giải pháp đề xuất giảm thiểu rủi ro: Đặt cờ android:debuggable thành false.

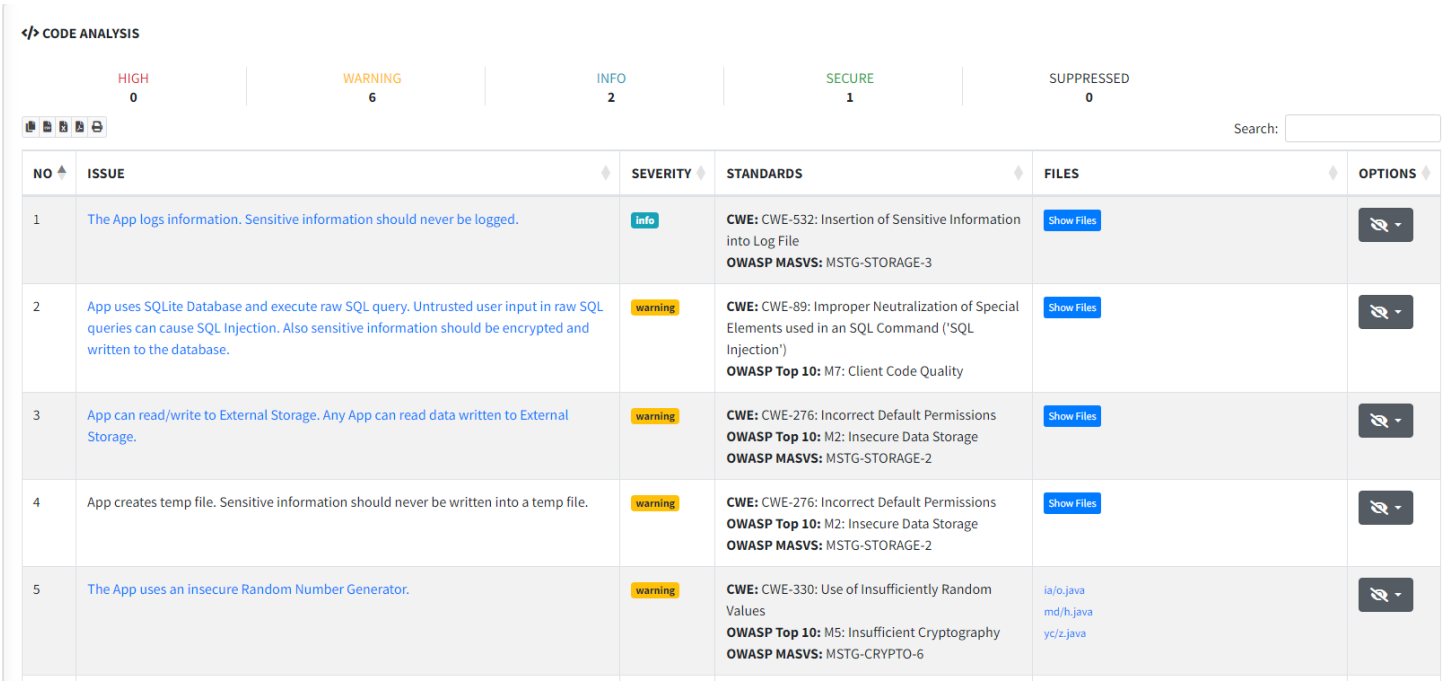
Sau khi hoàn thành rà quét và phân tích để đánh giá lỗ hổng mã nguồn trong

ứng dụng, kết quả và báo cáo chuyển đến đơn vị phát triển để khắc phục sự cố.



Hình 3.7. Thông tin phần mềm phiên bản 1.5.1

Phần mềm sau khi được cập nhật sau khi phát hiện lỗ hổng mã nguồn đã không còn cảnh báo mức cao.



Hình 3.8. Báo cáo phân tích mã nguồn của phần mềm phiên bản 1.5.1

3.4. Kết luận chương 3

Như vậy qua kết quả thực nghiệm, có thể thấy thấy phương pháp phát hiện lỗ hổng mã nguồn dựa trên tập luật bằng công cụ MobSF đã phát hiện được lỗ hổng mã nguồn đồng thời công cụ cũng cho đánh giá về mức độ cảnh báo nguy hại của lỗ hổng. Qua việc rà quét, phân tích dựa trên báo cáo xuất ra từ MobSF các lỗ hổng có cảnh báo mức cao đã được phát hiện. Từ đó đánh giá được mức rủi ro của lỗ hổng và đề xuất phương án, giải pháp khắc phục lỗ hổng cho đơn vị phát triển để cập nhật khắc phục tại phiên bản sau của phần mềm. Sau khi tiếp thu báo cáo phân tích và đề xuất, đơn vị phát triển đã cập nhật khắc phục lỗ hổng mức cảnh báo cao trong phiên bản sau của phần mềm.

KẾT LUẬN

Đề án đã trình bày một hướng tiếp cận về ứng dụng công cụ trong phát hiện lỗ hổng mã nguồn trên phần mềm nền tảng Android. Qua đó có thể đánh giá về việc sử dụng công cụ trong rà quét, phân tích, đánh giá, phát hiện các nguy cơ mã độc là một giải pháp khả thi, có hướng phát triển trong tương lai.

Đồng thời, đề án đã cho thấy hiệu quả của việc sử dụng công cụ để phát hiện lỗ hổng mã nguồn từ khả năng tự động hóa, phát hiện sớm trước khi rủi ro bị khai thác, tối ưu được việc sử dụng nhân lực trong khi vẫn đảm bảo được chất lượng của phần mềm.

Các kết quả đề án đạt được:

- Trình bày tổng quan về phát hiện lỗ hổng mã nguồn dựa trên tập luật.
- Trình bày về phương pháp phát hiện lỗ hổng mã nguồn dựa trên tập luật.
- Tiến hành thực nghiệm và đánh giá kết quả.

Hướng phát triển, nghiên cứu tiếp theo:

Mở rộng với hướng nghiên cứu phương pháp phát hiện lỗ hổng mã nguồn dựa trên tập luật với nền tảng IOS. Từ đó, tăng khả năng sử dụng công cụ để phát hiện, phân tích, đánh giá đa dạng và chính xác hơn.

DANH MỤC CÁC TÀI LIỆU THAM KHẢO

Tiếng Anh

[1] (Ajin Abraham, Magaofei, Matan Dobrushin, Vincent Nadal, 2023), *MobSF Documentation*, https://mobsf.github.io/docs. Truy cập ngày 1 tháng 10 năm 2023.

[2] Brian Chess, Jacob West (2010), Secure Programming with Static Analysis, *Software Security Series*, Pearson Education, Inc, United States, 47-70.

[3] (Common Weakness Enumeration, 2022), *CWE Documents* https://cwe.mitre.org/about/documents.html. Truy cập ngày 1 tháng 10 năm 2023.

[4] (Dave Dugal, Dale Rich, 2015), *CVSS v3.0 Specification,* https://www.first.org/cvss/v4-0/cvss-v40-presentation.pdf. Truy cập ngày 1 tháng 10 năm 2023.

[5] Diaeddin Rimawi, Samer Zein (2020), "A Static Analysis of Android Source Code for Design Patterns Usage", *International Journal of Advanced Trends in Computer Science and Engineering*, (9)2, 2178-2186.

[6] Janaka Senanayake, Harsha Kalutarage, Uk Mhd Omar Al-kadri, Andrei Petrovski, Luca Piras (2023), "Android Source Code Vulnerability Detection: A Systematic Literature Review", *ACM Computing Surveys*, (55)9, 1-29.

[7] Mamoona Humayun, Mahmood Niazi, Noor Zaman Jhanjhi, Sajjad Mahmood (2022), Toward a readiness model for secure software coding, *Software: Pract Exper*, (53), 1013–1035.

[8] (OWASP, 2019), *Application Security Verification Standard 4.0*, Creative Commons Attribution ShareAlike 3.0 license, 7-80.