TẬP ĐOÀN CÔNG NGHIỆP - VIỄN THÔNG QUÂN ĐỘI
TỔNG CÔNG TY GIẢI PHÁP DOANH NGHIỆP VIETTEL

VIETTEL DIGITAL TALENT CLOUD 2024

ĐỀ TÀI NGHIÊN CỨU: Bảo mật trong Docker Container

Thực tập sinh: Đinh Việt Quang

Ha Noi, May - 2024

Contents

Abstract

Docker luôn là một công cụ được trong cộng đồng phát triển phần mềm nhờ tính linh hoạt, di động và khả năng mở rộng của các container. Tuy nhiên, những lo ngại về lỗ hổng bảo mật đã gia tăng khi bảo mật của các ứng dụng ngày càng phụ thuộc vào sự an toàn của các image đóng vai trò là các khối khởi tạo của ứng dụng. Khi ngày càng nhiều quy trình phát triển chuyển sang điện toán đám mây, việc xác thực bảo mật của các image được lấy về từ các kho lưu trữ khác nhau trở nên vô cùng quan trọng. Một hệ thống tích hợp và triển khai liên tục (CI/CD) xác thực bảo mật của các Docker image trong suốt vòng đời phát triển phần mềm được đề xuất nhằm làm rõ lỗ hổng bảo mật và ước tính độ hiệu quả trong việc xác định các lỗ hổng này. Hơn nữa, ta có thể phân tích để đánh giá bảo mật của các Docker container dựa trên hành vi của chúng.

1 Mở đầu

Các container đã đạt được sức hút đáng kể trong cộng đồng phát triển phần mềm vì chúng cho phép các nhà phát triển tránh được việc cấu hình thư viện và các phụ thuộc tốn thời gian. Một image là một tệp chứa mã nguồn, cấu hình và thư viện cần thiết để chạy một ứng dụng. Các container là các phiên bản của những image này, với mỗi phiên bản có cùng các phụ thuộc cơ bản. Vì chúng chỉ bao gồm các hướng dẫn và mã nguồn cần thiết để ứng dụng chạy, các container nhẹ hơn so với các phương pháp thay thế như máy ảo (VM). Nhiều container có thể chạy trên một máy vật lý hoặc máy ảo duy nhất, làm cho chúng trở nên lý tưởng cho nhiều giai đoạn của chu kỳ phát triển phần mềm.

Microservice là một mô hình kiến trúc phần mềm thường được sử dụng với các container. Microservice nhằm mục đích làm cho việc phát triển phần mềm dễ dàng hơn thông qua việc sử dụng tối ưu các tài nguyên khác nhau. Về bản chất, mỗi chức năng phần mềm được cung cấp trong một thành phần ứng dụng đơn lẻ. Các thành phần sau đó sử dụng API để tương tác với nhau. Microservice đóng vai trò là các khối xây dựng để thiết kế và phát triển phần mềm có khả năng mở rộng và dễ bảo trì. Cách tiếp cận này giảm thiểu sự phụ thuộc giữa các chức năng phần mềm chính và tối thiểu hóa mã nguồn mà các nhà phát triển phải tương tác.

Docker là một nền tảng dưới dạng dịch vụ (PaaS) phổ biến cho các container. Nhiều tổ chức như Google và Amazon đang tích hợp Docker vào vòng đời phát triển phần mềm của họ, vì Docker cho phép các nhà phát triển nhanh chóng thiết kế, phát triển, kiểm thử và triển khai ứng dụng trong khi tối ưu hóa việc sử dụng tài nguyên [3]. Do sự phổ biến của Docker, cộng đồng phát triển đã tạo ra các kho lưu trữ Docker Image như Docker Hub, để tăng khả năng tái sử dụng và khuyến khích việc chia sẻ tệp. Với mô hình do các container và microservice mang lại, phát triển phần mềm ngày càng phụ thuộc vào các thành phần nhỏ, có thể tái sử dụng được phát triển độc lập và phân phối bởi các tổ chức khác nhau. Tuy nhiên, sự phụ thuộc này đang làm dấy lên những lo ngại về bảo mật của toàn bộ chuỗi phân phối Docker Image. Các kiến trúc sư tại Docker hiện khuyến khích các nhà phát triển và nhà xuất bản bao gồm phân tích rủi ro mà coi toàn bộ chuỗi phân phối như một thực thể có thể gây hại [3]. Cơ chế bảo mật nhiều lớp ở cấp độ kho lưu trữ Docker có thể giảm thiểu các lỗ hổng bảo mật bị đưa vào các kho lưu trữ Docker.

Trong nghiên cứu, công việc được thiết kế để cải thiện các biện pháp bảo mật khi sử dụng container như một phần của quá trình phát triển phần mềm. Cụ thể, một quy trình CI/CD được triển khai để đánh giá độ bảo mật của Docker Image. Hơn nữa, nó còn được sử dụng để chỉ ra những nội dung độc hại và chỉ ra cách phát hiện các hành vi bất thường.

2 Thiết kế

Một phương pháp bảo mật nhiều lớp cho CI/CD trong quá trình phát triển các Docker Image được đề xuất. Thiết kế này cho phép các nhà phát triển linh hoạt tích hợp các công cụ phân tích động (Dynamic Analysis) bảo mật mong muốn vào quy trình phát triển của họ. Ngoài ra, nó khuyến khích việc định nghĩa các chính sách bảo mật rõ ràng cho các Docker image được sử dụng trong môi trường phát triển của họ.

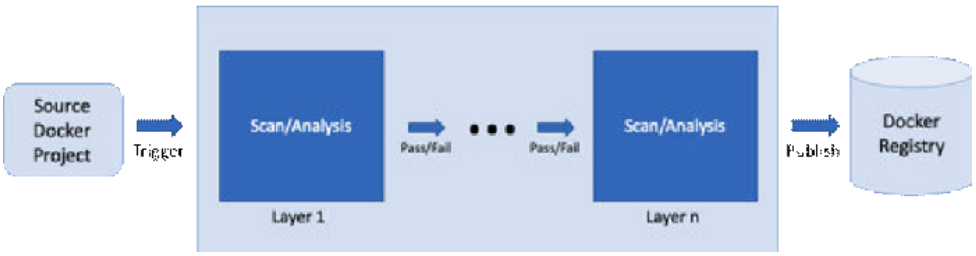

Figure 1: Luồng CI/CD cho bảo mật Docker Image

Hình 1 minh họa luồng công việc của CI/CD dành cho các Docker Image. Dự án Docker nguồn khởi tạo quá trình CI/CD. Mỗi quy trình có thể chứa một số lượng tùy ý các giai đoạn với các quét lỗ hổng và Static hoặc Dynamic Analysis. Khi mã nguồn của một dự án Docker mới đi vào quy trình và trên tất cả các cập nhật tương lai đối với mã nguồn của dự án, nó sẽ kích hoạt các kiểm tra bảo mật được chỉ định. Để cung cấp cho các nhà phát triển và quản trị viên sự linh hoạt trong việc mở rộng, mỗi lớp có thể thực thi các kiểm tra bảo mật của nó theo cách tùy ý. Chỉ khi dự án Docker nguồn vượt qua các bài kiểm tra bảo mật ở mỗi lớp, dự án mới được xuất bản lên kho lưu trữ Docker.

Do nguy cơ tăng cao khi triển khai sử dụng Dynamic Analysis trên cùng một hệ thống với quy trình, Ta đề xuất một công cụ thứ hai để đạt được mục tiêu này. Một API được triển khai để tự động hóa và thu thập thông tin hữu ích về các Docker Image được chỉ định nhằm xác định lỗi và nội dung độc hại. Dịch vụ API này có hai mục đích: đầu tiên, nó là một công cụ nghiên cứu cho các nhà phát triển để phân tích các image công khai; thứ hai, nó là một lớp thứ hai trong cơ chế bảo mật nhiều lớp. Dịch vụ này cho phép các nhà phát triển phân tích bất kỳ image công khai nào để tìm lỗ hổng và phần mềm độc hại, sử dụng Clair cho phân tích lỗ hổng và quét nội dung độc hại bằng VirusTotal. Cuối cùng, nó thực hiện Dynamic Analysis cơ bản bằng cách chạy Image trong một khoảng thời gian để thu thập các thay đổi tệp, lưu lượng mạng, và danh sách các tiến trình trong suốt quá trình thực thi của Image.

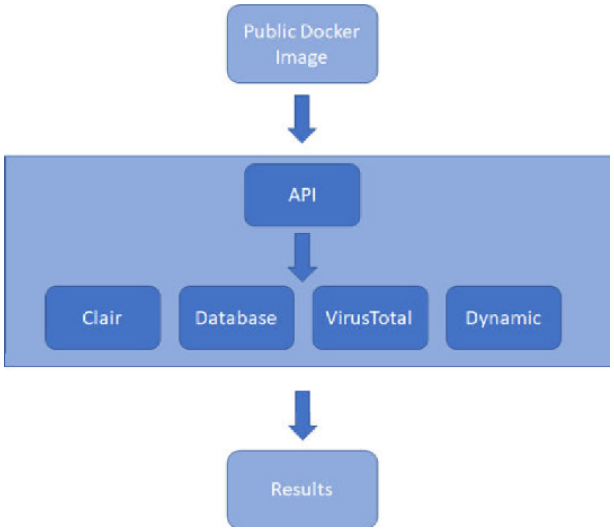

Figure 2: Luồng dịch vụ cho nghiên cứu các Image công khai

Các Image độc hại thường bao gồm các tập lệnh bash tạo các SSH trở lại máy chủ điều khiển và điều khiển khi container chạy. Các Image độc hại này cố gắng tải xuống các tệp nhị phân bổ sung và cài đặt mã shell. Tuy nhiên, chỉ có Dynamic Analysis mới nắm bắt được hành vi này. Để giải quyết vấn đề này, một dịch vụ phụ trong công cụ đã được phát triển để tự động hóa việc kiểm tra động các Docker Image. Quy trình này sử dụng Docker-in-Docker (dind) để tự động hóa và thu thập hành vi này trong một sandbox. Mặc dù Docker-in-Docker làm việc cách ly, nó vẫn được thực thi ở chế độ đặc quyền và vẫn có quyền truy cập vào máy chủ và có các quyền tương tự như Docker chạy trên máy chủ. Do đó, điều quan trọng là phải chạy sandbox Docker-in-Docker này trong một hypervisor hoặc dịch vụ đám mây cung cấp mức độ cách ly thích hợp.

3 Triển khai

Trong mục này, AWS được sử dụng để triển khai hệ thống.

Một quy trình hai giai đoạn đã được triển khai để minh họa cho CI/CD. Clair và Anchore Engine cung cấp các phân tích bảo mật. Hình 3 minh họa kiến trúc AWS. Một Đám Mây Riêng Ảo (VPC) cô lập hợp lý các tài nguyên khác nhau trong hệ sinh thái AWS. Clair và Anchore Engine được cô lập thêm bằng cách sử dụng hai vùng khả dụng. Mỗi vùng khả dụng có một subnet công cộng và một subnet riêng tư, và trong mỗi subnet công cộng, cổng kết nối cho phép các thành phần trong subnet riêng tư truy cập Internet. Một cụm cơ sở dữ liệu PostgreSQL trải dài qua hai subnet riêng tư.

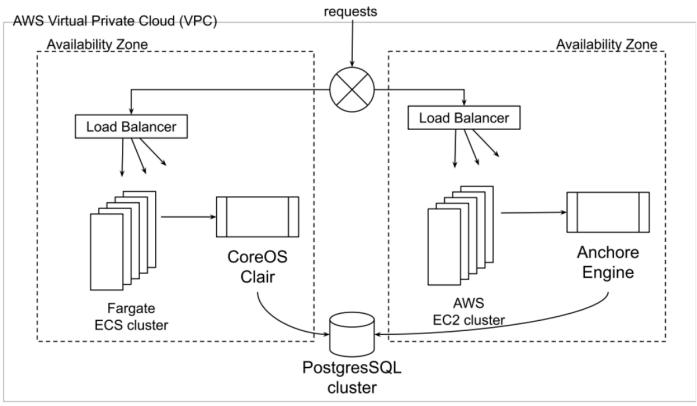

Figure 3: Kiến trúc AWS cho dịch vụ backend

Để lưu trữ nội dung nguồn của dự án Docker, ta sử dụng một kho lưu trữ AWS CodeCommit. Mỗi khi trạng thái của kho lưu trữ thay đổi, sự kiện sẽ kích hoạt việc thực thi các lớp bảo mật trong CI/CD.

Ta thực hiện quét lỗ hổng trong lớp đầu tiên bằng cách sử dụng CoreOS Clair. Clair tích hợp liền mạch với quy trình bằng cách cung cấp các điểm cuối API nơi các ứng dụng khách có thể chạy quét lỗ hổng. Clair phân loại CVE thành bảy loại: không xác định, không đáng kể, thấp, trung bình, cao, nghiêm trọng và defcon1, với không xác định là ít nghiêm trọng nhất và defcon1 là nghiêm trọng nhất. Hai giá trị cấu hình, ngưỡng loại và số lượng lỗ hổng, xác định xem một Image có vượt qua hay không kiểm tra CVE. Nếu quá trình quét phát hiện các lỗ hổng nghiêm trọng hơn ngưỡng loại và số lượng lỗ hổng vượt quá ngưỡng số lượng, thì giai đoạn CI/CD sẽ thất bại trên Image. Một vùng khả dụng AWS chạy các dịch vụ liên quan đến Clair với một cụm Fargate Elastic Container Service (ECS) trong subnet riêng tư. Một bộ cân bằng tải phân phối các yêu cầu trên cụm ECS.

Nếu dự án Docker nguồn vượt qua phân tích bảo mật của lớp đầu tiên, nó sẽ trải qua các bước tương tự trong lớp thứ hai bằng cách sử dụng Anchore Engine. Mặc dù một chính sách bảo mật tùy chỉnh có thể được định nghĩa, nguyên mẫu trong nghiên cứu sử dụng chính sách bảo mật mặc định của Anchore, thực hiện các kiểm tra lỗ hổng nhẹ (ví dụ: Image không chứa các gói có lỗ hổng nghiêm trọng) và kiểm tra Dockerfile (ví dụ: cổng 22 không nên được mở). Mặc dù Anchore Engine cũng thực hiện báo cáo dựa trên CVE, ta có thể thấy rằng chỉ dựa vào báo cáo dựa trên CVE là không đủ để thực hiện toàn bộ quy trình CI/CD. Do tính chất microservice của cách triển khai Anchore, một cụm Fargate ECS không khả thi. Do đó, ta sẽ sử dụng một cụm ECS có khả năng khởi chạy các phiên bản Elastic Compute Cloud (EC2) c4.xlarge và lưu trữ các dịch vụ microservice của Anchore Engine.

Nếu một Image vượt qua tất cả các lớp bảo mật, Image sẽ được xuất bản vào một kho lưu trữ AWS Elastic Container Registry (ECR) chuyên dụng, nơi nó có sẵn cho những người khác có quyền truy cập vào kho lưu trữ đó. Hoạt động này được thực hiện trong lớp cuối cùng của quy trình CI/CD.

Việc triển khai cải thiện đáng kể [1] so với các công việc trước đây. Thứ nhất, nhiều lớp được tích hợp vào CI/CD. Thứ hai, nó yêu cầu thiết lập thủ công ít hơn đáng kể: ví dụ, các nhà phát triển không cần phải tạo một kho lưu trữ ECR cho mỗi phân tích bảo mật [1]. Sử dụng việc triển khai này, các nhà phát triển và quản trị viên có thể cung cấp tất cả các dịch vụ và tài nguyên cần thiết cho quy trình CI/CD của họ bằng cách sử dụng một mẫu AWS CloudFormation duy nhất. Ngoài ra, các bộ cân bằng tải sử dụng tự động mở rộng để hệ thống mở rộng theo chiều ngang dựa trên tải.

Một cách đơn giản để xác định xem một Image có chứa mã độc hại hay không là sử dụng một công cụ quét virus. Ta sử dụng VirusTotal vì số lượng lớn các công cụ quét virus mà nó hỗ trợ. Mặt hạn chế của việc phụ thuộc vào phần mềm diệt virus là hầu hết các công cụ quét virus chỉ thực hiện phân Static Analysis và tìm kiếm các chữ ký trong khi một số công cụ quét virus thực hiện Dynamic Analysis với các bộ mô phỏng riêng của nó. Tuy nhiên, gần đây có các bằng chứng cho thấy mã độc hại đang bắt đầu tránh được các bộ mô phỏng của phần mềm diệt virus [2]. Bằng cách chạy Image động trong Docker-in-Docker, Image đang chạy trong chế độ đặc quyền nên không có bộ mô phỏng nào. Image về cơ bản đang chạy như là nó trên máy chủ, điều này nên cho phép phân tích chính xác hơn.

Để bắt gói tin mạng của một Image, một container khác chứa tcpdump chạy song song với container mục tiêu. Hai container này chia sẻ cài đặt mạng. Bắt đầu container tcpdump yêu cầu vài giây trước khi nó bắt gói tin mạng, vì vậy bất kỳ giao thông ban đầu nào cũng có thể bị mất. Có một số trường hợp mà container quan tâm không chạy đủ lâu cho container tcpdump bắt đầu. Trong trường hợp của các Image độc hại, các container thường chạy trong khoảng thời gian dài hơn vì họ cần giao tiếp lại với máy chủ điều khiển để nhận các hướng dẫn tiếp theo. Ta lưu giao thông mạng trong một tệp bắt gói (pcap) và đẩy nó lên Dịch vụ Lưu trữ Đơn giản (S3) để phân tích sau này.

Docker có tùy chọn ghi lại các thay đổi hệ thống tệp, mà cũng được sử dụng trong Dynamic Analysis của mình. Sự khác biệt là các thay đổi được thực hiện trên Image cơ bản từ khi container đang chạy. Nhật ký này hữu ích khi phân tích các Image độc hại tải xuống và thực thi các tệp nhị phân mới không phải là phần của Image gốc. Một khả năng phân tích khác là liệt kê tất cả các tiến trình trong container. Phần động của API thực hiện một danh sách các tiến trình mỗi giây trong suốt quá trình chạy.

4 Đánh giá

4.1 Kết quả thực nghiệm

Nghiên cứu sử dụng hai dự án Docker để đánh giá quy trình bảo mật CI/CD này (xem Bảng 1). Một dự án, một trang web nginx5 mà mở cổng 80 và 443 cho lưu lượng HTTP và HTTPS, giống như đã được sử dụng trong các công việc trước đây [1]. PostgreSQL 11 được làm dự án Docker thứ hai vì Image sẽ tương đối an toàn trước các lỗ hổng. Đối với CoreOS Clair, một Image được thông qua nếu và chỉ nếu, Image không phát hiện ra năm mươi hoặc nhiều lỗ hổng được phân loại là “cao”. Image trong Hình 4 hiển thị số lượng lỗ hổng được tạo ra bởi CoreOS Clair trong CI/CD được đề xuất. Kết quả xác nhận dự đoán về sự an toàn của PostgreSQL 11: Image không được phát hiện ra bất kỳ lỗ hổng nào. Tuy nhiên, Image trang website nginx đã được phát hiện ra 143 lỗ hổng tổng cộng. Ngay cả với tất cả các lỗ hổng này, Image vẫn thông qua vì nó không vượt quá ngưỡng của năm mươi hoặc nhiều hơn lỗ hổng được phân loại là “cao” hoặc nghiêm trọng hơn.

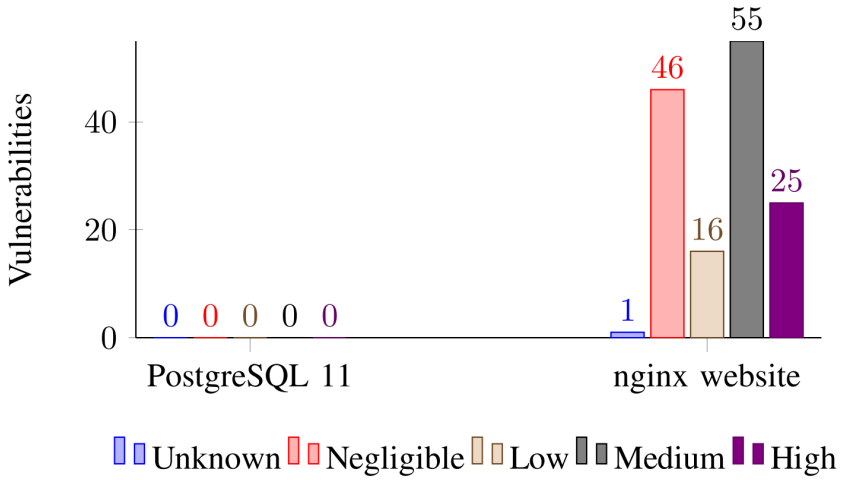

Figure 4: Kết quả quét lỗ hổng Clair

Lớp thứ hai của quy trình thực hiện chính sách bảo mật mặc định của Anchore Engine, bao gồm các kiểm tra lỗ hổng nhẹ và kiểm tra Dockerfile. PostgreSQL 11 Image đáp ứng tất cả các yêu cầu bảo mật của chính sách bảo mật mặc định và đã được xuất bản vào kho lưu trữ Docker. Ngược lại, trang web nginx mẫu không vượt qua chính sách bảo mật và không được xuất bản vào kho lưu trữ.

Image	Tag
postgres	11.5
ubuntu	18.04
python	3.6-alpine
node	10.16.0-alpine
nginx	1.17.2
buamod/eicar	latest
zoolu2/jauto	latest

Table 1: Danh sách Docker images được sử dụng trong đánh giá.

Clair không phát hiện ra bất kỳ lỗ hổng nào cho các Image được liệt kê trong Bảng 1, điều này được dự kiến vì các Image là các Image mới nhất có sẵn. Không mong đợi có lỗ hổng trừ khi nhà phát triển thêm một gói cũ vào Image cơ sở. Ngoài ra, Dynamic Analysis cũng không tập trung vào việc tìm lỗ hổng.

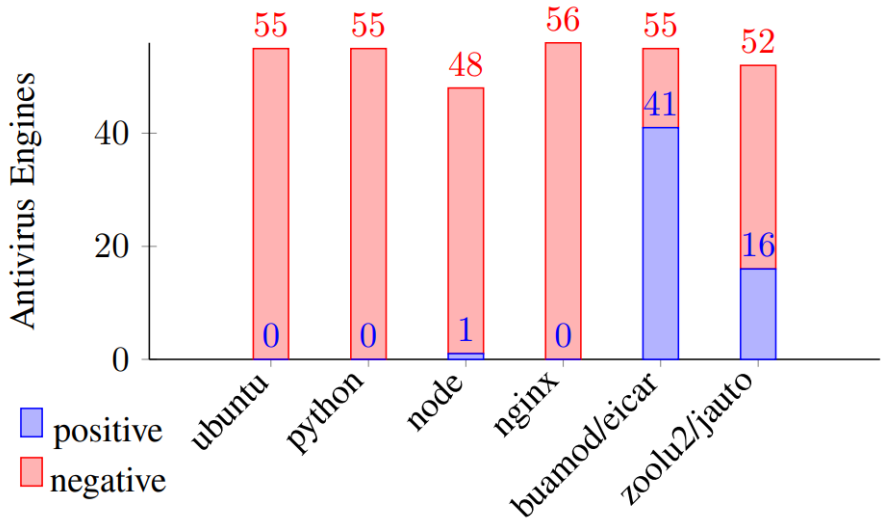

Figure 5: Kết quả quét VirusTotal. Kết quả tích cực cho thấy phần mềm chống virus cho thấy Image có chứa nội dung độc hại.

Kết quả của VirusTotal, được tóm tắt trong hình 5, cũng không quá bất ngờ. Các ubuntu, python, node và nginx Image được hỗ trợ chính thức bởi Docker Hub và do đó có khả năng đã được phân tích để phát hiện phần mềm độc hại trước khi được xuất bản. Ta không chắc chắn tại sao node đã được một nhà cung cấp phần mềm diệt virus đánh dấu, nhưng mình có thể tin rằng đó là một kết quả không tích cực. Hai Image cuối cùng, buamod/eicar và zoolu2/jauto, được dự đoán là dương tính do có nội dung độc hại được biết trước. Buamod/eicar Image chứa một tệp có chữ ký nhị phân được sử dụng để kiểm tra phần mềm diệt virus, và Image zoolu2/jauto chứa các tập lệnh tải xuống và cài đặt một trình đào tiền điện tử.

Khi kiểm tra các Image để thử nghiệm bằng Dynamic Analysis trong công cụ API, kết quả không đáng chú ý cho các Image ubuntu, python và node. Thảo luận tiếp theo tập trung vào các Image nginx và zoolu2/jauto. Hình 6 tóm tắt số lượng sửa đổi tệp, tiến trình đang chạy và các truy vấn Hệ thống Tên Miền (DNS) cho mỗi Image sau 30 giây thực thi.

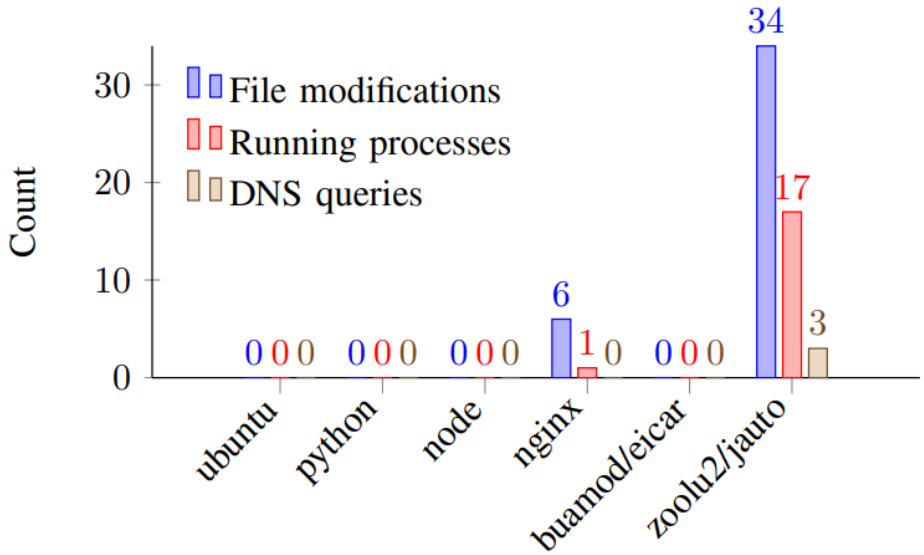

Figure 6: Kết quả khi sử dụng Dynamic Analysis API.

Các thay đổi hệ thống tệp liên quan đến chức năng của Image và các hành động mà Image đang cố gắng thực hiện. Điều này đặc biệt hữu ích khi kiểm tra các tệp độc hại và trong trường hợp của các Image thử nghiệm. Ví dụ, zoolu2/jauto Image viết hai tập lệnh, cmd.sh và cmd1.sh, trong /root. Image cũng khảo sát hệ thống và lưu thông tin như siêu dữ liệu CPU. Image này cũng thực hiện một số tìm kiếm IP và quét bằng cách sử dụng Shodan, một công cụ tìm kiếm cho các thiết bị kết nối Internet. Một cờ đỏ lớn xuất hiện khi Image khởi chạy các dịch vụ SSH và Tor. Trong khi đó, các thay đổi tệp của nginx bao gồm các tệp tạm thời, tất cả đều nằm trong /var/cache/nginx.

Hơn nữa, trình quét API động cũng tải lên một tệp pcap của phân tích mạng trong khi Image đang chạy. Nhiều Image không có bất kỳ giao thông mạng nào và do đó có các tệp pcap trống. Từ các bản ghi DNS được thu thập trong phân tích này, chúng ta có thể biết rằng container zoolu2/jauto xác thực với shodan.io để thực hiện các tìm kiếm địa chỉ IP để tìm cổng mở và dịch vụ đang chạy trên máy chủ. Phát hiện đáng chú ý tiếp theo là yêu cầu DNS cho us-east.cryptonight-hub.miningpoolhub.com, một nơi cho các trình đào tiền điện tử phân bổ tài nguyên khai thác. Mặc dù không phân tích hoàn toàn tất cả các lưu lượng mạng, nhưng bằng chứng cho thấy rằng Image này độc hại và đang cố gắng khai thác tiền điện tử. Phân tích cho thấy rằng Image này không nên được tin cậy để thực thi bên ngoài một môi trường cát.

4.2 Hạn chế

Kế hoạch hiện tại dựa nhiều vào số lượng lỗ hổng được quét. Mặc dù quyết định này có thể phục vụ như là một giai đoạn ban đầu tốt cho các quy trình DevSecOps, nhưng một số lượng lỗ hổng tùy ý không đủ cho các yêu cầu bảo mật cấp sản xuất. Ngoài ra, chính sách bảo mật mặc định của Anchore được sử dụng trong thí nghiệm có thể không xác định tất cả các yêu cầu bảo mật mà tổ chức phải đáp ứng.

Dịch vụ API cho Dynamic Analysis hiện tại hỗ trợ việc quét các Image công cộng nhưng không hỗ trợ tải lên Dockerfile để phân tích. Các Image tùy chỉnh thường được thiết kế bởi nhà phát triển và do đó nội dung của Dockerfile nên được biết đến. Tuy nhiên, có những trường hợp mà các thư viện và phụ thuộc chính thức có thể bị sửa đổi, gây nguy hiểm cho tổng thể bảo mật của Image. Việc mở rộng dịch vụ API để bao gồm việc quét các Image tùy chỉnh có thể được cân nhắc. Trong những trường hợp này, phần Dynamic Analysis của dịch vụ API có thể giúp giám sát và phát hiện các trường hợp như vậy. Từ kết quả trên, rõ ràng rằng các Image Docker an toàn nên ít nhất qua một quét virus và thực hiện ít sửa đổi trên hệ thống tệp.

5 Kết luận

Các nhà phát triển sử dụng các container hiện đang dễ bị tổn thương bởi phần mềm độc hại và thiếu các công cụ có khả năng đo lường rủi ro này một cách hiệu quả. Các công cụ hiện có, mặc dù hiệu quả, nhưng tốn thời gian và khó khăn để triển khai và có thể đưa ra các rủi ro mới nếu không triển khai đúng cách. Công việc đã giải quyết những vấn đề này bằng cách tạo ra các công cụ dễ sử dụng để phát hiện lỗ hổng và mã độc hại.

Kết quả cho thấy rằng quét virus và Dynamic Analysis là hiệu quả trong việc phát hiện hành vi độc hại trong các Docker container. Cụ thể, các Image an toàn thường chỉ có vài sửa đổi tệp, tiến trình đang chạy và truy vấn DNS trong khi các Image độc hại thường tải xuống và thực thi các tệp không xuất hiện ban đầu trong Image. Bằng cách sử dụng dịch vụ API cho Dynamic Analysis, các nhà phát triển được trang bị tốt hơn để đưa ra quyết định về việc sử dụng các

Image cơ sở nào. Tự động hóa các kiểm tra tĩnh và thời gian chạy cũng giúp các nhà phát triển xây dựng các ứng dụng an toàn hơn.

Công việc của tương lai có thể là Dynamic Analysis được vào quy trình CI/CD, điều này đòi hỏi tạo ra một hộp cát cho các Image Docker để nó không ảnh hưởng đến các dịch vụ hoặc tài nguyên khác. Khám phá cách sử dụng Anchore Engine để thực hiện Dynamic Analysis trong quy trình CI/CD có thể đáng lưu tâm. Một phần mở rộng thú vị khác là áp dụng học máy vào phần Dynamic Analysis của dịch vụ API. Thay vì người dùng phải xem xét thủ công các thay đổi hệ thống tệp và nhật ký lưu lượng mạng mà API xuất ra, học máy tự động hóa việc phân loại các Image là độc hại hoặc lành tính dựa trên hoạt động của chúng. Các sửa đổi tệp, tiến trình đang chạy và các truy vấn DNS là một điểm khởi đầu lý tưởng cho các chữ ký

References

[1] V Adethyaa and T Jernigan. Scanning docker images for vulnerabilities using clair amazon ecs ecr and aws codepipeline. *AWS Compute Blog*, 2018.

[2] Jeremy Blackthorne, Alexei Bulazel, Andrew Fasano, Patrick Biernat, and Bülent Yener. {AVLeak}: Fingerprinting antivirus emulators through {Black-Box} testing. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[3] Steafan Winkle. Security assurance of docker containers. 2020.