

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TPHCM

KHOA CÔNG NGHỆ THÔNG TIN



NGÀNH: AN TOÀN THÔNG TIN

BÁO CÁO CUỐI KỲ

MÔN: PHÁP LÝ KỸ THUẬT SỐ

**XÂY DỰNG PLAYBOOK HỖ TRỢ ĐIỀU TRA PHÁP
CHỨNG CHO HỆ THỐNG CONTAINERIZED APPLICATIONS
TRONG MÔI TRƯỜNG DEVOPS**

Sinh viên thực hiện: Nhóm 14

- Nguyễn Thắng Lợi, 22162023
- Lê Anh Khoa, 22162016

TP. Hồ Chí Minh, tháng 10 năm 2025

MỤC LỤC

PHẦN MỘT: MỞ ĐẦU.....	4
1. Giới thiệu đề tài.....	4
2. Mục tiêu nghiên cứu.....	4
3. Phương pháp và phạm vi nghiên cứu.....	5
PHẦN HAI: NỘI DUNG.....	6
CHƯƠNG 1: TỔNG QUAN VỀ HỆ THỐNG CONTAINERIZED APPLICATIONS VÀ CÁC THÁCH THỨC TRONG ĐIỀU TRA SỐ.....	6
1.1. Tổng quan về hệ thống containerized applications.....	6
1.1.1. Khái niệm về containerized application.....	6
1.1.2. Đặc điểm của hệ thống containerized applications.....	7
1.2. Các thách thức trong điều tra số cho hệ thống containerized applications.....	8
1.2.1. Tính tạm thời.....	8
1.2.2. Biến động dữ liệu.....	8
1.2.3. Multi-tenancy và quyền riêng tư.....	9
1.2.4. Phụ thuộc vào nhà cung cấp và quyền truy cập hạn chế.....	9
1.2.5. Khối lượng dữ liệu lớn và quản lý nhật ký.....	10
1.2.6. Mã hóa và kiểm soát truy cập.....	10
1.2.7. Chain of Custody.....	11
1.2.8. Hạn chế về công cụ và kỹ thuật.....	11
1.3. Đề xuất quy trình thu thập và phân tích.....	11
1.3.1. Thiết lập pipeline.....	12
1.3.2. Quy trình thu thập khi xảy ra sự cố.....	12
1.3.3. Cách thức phân tích các loại bằng chứng.....	13
1.3.4. Xây dựng timeline.....	14
1.3.5. Điều tra nguyên nhân / vị trí sự cố.....	14
1.3.6. Quy trình khắc phục.....	15
CHƯƠNG 2: XÂY DỰNG PIPELINE TỰ ĐỘNG HÓA VIỆC THU THẬP CÁC ARTIFACTS TỪ ORCHESTRATION.....	17
2.1. Xác định các nguồn artifacts quan trọng.....	17
2.2. Triển khai hệ thống Kubernetes với MicroK8s.....	18
2.3. Cấu hình việc ghi logs của MicroK8s.....	21
2.4. Triển khai Fluent Bit và Suricata để thu thập và gửi logs.....	23
2.5. Thiếp lập Splunk cho việc theo dõi và phân tích logs.....	27
CHƯƠNG 3: THỰC NGHIỆM MÔ PHỎNG TÂN CÔNG VÀ ĐÁNH GIÁ.....	29
3.1. Mô phỏng tấn công vào ứng dụng web.....	29
3.2. Đánh giá hiệu quả của pipeline và quy trình được đề xuất.....	30
PHẦN BA: KẾT LUẬN.....	32
1. Các kết quả đạt được.....	32
2. Những hạn chế.....	33
3. Hướng phát triển.....	33
TÀI LIỆU THAM KHẢO.....	35

PHẦN MỘT: MỞ ĐẦU

1. Giới thiệu đề tài

Việc áp dụng containerization và kiến trúc microservices đã trở nên phổ biến nhờ những lợi ích thiết thực: chúng cho phép triển khai nhanh hơn, mở rộng quy mô linh hoạt, sử dụng tài nguyên hiệu quả và cô lập lỗi vượt trội so với các hệ thống nguyên khói truyền thống. Tuy nhiên, chính sự dịch chuyển sang các ứng dụng được container hóa và mô hình DevOps lại đang tạo ra những thách thức nghiêm trọng cho lĩnh vực điều tra pháp chứng kỹ thuật số và ứng cứu sự cố. Các quy trình và bộ công cụ điều tra truyền thống tỏ ra không còn đủ khả năng thích ứng với bản chất phức tạp và liên tục biến động của những môi trường này.

Thách thức cốt lõi đến từ tính tạm bợ của container. Vòng đời của chúng từ lúc khởi tạo, vận hành đến khi bị hủy có thể chỉ diễn ra trong vài giây, khiến các bằng chứng số quan trọng bị mất mát gần như tức thời. Khi một sự cố xảy ra, việc thiếu một quy trình thu thập được chuẩn bị trước sẽ phá vỡ chain of custody, khiến mọi dữ liệu thu được sau đó mất đi giá trị pháp lý.

Vấn đề không chỉ dừng lại ở đó. Môi trường container vốn đã phức tạp với nhiều lớp tương tác chồng chéo, từ hệ thống điều phối (Orchestration), quy trình CI/CD, các registry, cho đến container runtime. Một sự cố bảo mật hay một cuộc tấn công có thể để lại dấu vết ở bất kỳ điểm nào trong chuỗi cung ứng này, từ giai đoạn xây dựng image cho đến khi triển khai vận hành.

Đề tài “*Xây Dựng Playbook Hỗ Trợ Điều Tra Pháp Chứng Cho Hệ Thống Containerized Applications Trong Môi Trường Devops*” được phát triển nhằm nhằm đề xuất một pipeline tự động hóa tác vụ thu thập bằng chứng và quy trình phân tích cụ thể để hỗ trợ điều tra trong các môi trường đặc thù này.

2. Mục tiêu nghiên cứu

- Xây dựng pipeline thu thập chứng cứ tự động đã được xây dựng và tối ưu hóa cho môi trường DevOps. Yêu cầu thiết kế là phải đảm bảo tuyệt đối tính toàn vẹn và bất biến của dữ liệu pháp chứng trong suốt quá trình thu thập.

- Tích hợp các công cụ thu thập chứng cứ trực tiếp. Dữ liệu thu thập được sẽ tự động lưu trữ tập trung vào một kho bảo quản an toàn, giúp các điều tra viên truy cập và tiến hành phân tích bằng chứng một cách hiệu quả và đáng tin cậy.

3. Phương pháp và phạm vi nghiên cứu

- Phương pháp lý thuyết: phân tích và tổng hợp các tài liệu khoa học, tài liệu kỹ thuật liên quan đến môi trường DevOps và Containerized Applications, làm rõ các thách thức đặc thù trong việc thu thập, quản lý và lưu trữ log từ các hệ thống có tính phân tán và vòng đời ngắn.

- Phương pháp thực nghiệm: xây dựng các kịch bản sự cố mô phỏng các tình huống tấn công. Thông qua các kịch bản này, hệ thống được đánh giá về khả năng thu thập và, quan trọng nhất, là khả năng đảm bảo tính toàn vẹn của dữ liệu pháp chứng trong suốt quá trình xử lý.

PHẦN HAI: NỘI DUNG

CHƯƠNG 1: TỔNG QUAN VỀ HỆ THỐNG CONTAINERIZED APPLICATIONS VÀ CÁC THÁCH THỨC TRONG ĐIỀU TRA SỐ

1.1. Tổng quan về hệ thống containerized applications

Việc container hóa ứng dụng, chủ yếu dựa trên các công nghệ như Docker và Kubernetes, cho phép nhà phát triển đóng gói phần mềm vào các môi trường độc lập và có thể tái lập. Điểm mấu chốt của phương pháp này là đảm bảo tính nhất quán vận hành khi ứng dụng di chuyển qua các môi trường khác nhau, từ máy trạm cá nhân đến hệ thống máy chủ sản xuất. Các khái niệm nền tảng và đặc điểm của kiến trúc này chính là nội dung sẽ được phân tích sau đây.

1.1.1. Khái niệm về containerized application

Về bản chất, một ứng dụng container hóa chính là phần mềm được đóng gói (encapsulated) trọn vẹn bên trong một container. Container có thể được hiểu là một môi trường thực thi (runtime) độc lập và gọn nhẹ, chứa đựng mã nguồn ứng dụng cùng toàn bộ các thành phần phụ thuộc (dependencies) của nó, bao gồm thư viện, tệp nhị phân, tệp cấu hình và các yếu tố runtime khác. Sự khác biệt căn bản của container so với máy ảo (Virtual Machines), vốn giả lập toàn bộ một hệ điều hành, nằm ở việc container chia sẻ chung nhân (kernel) của hệ điều hành máy chủ (host OS). Kiến trúc này giúp chúng hiệu quả hơn đáng kể về tài nguyên (CPU, bộ nhớ) và khởi động nhanh vượt trội (thường tính bằng giây, so với hàng phút của VM).

Chính quá trình đóng gói này đảm bảo ứng dụng chạy nhất quán trên mọi môi trường, từ máy tính cá nhân của lập trình viên (development laptops) đến môi trường production trên đám mây. Đây cũng là giải pháp triệt để cho vấn đề kinh điển "it works on my machine" (trên máy tôi chạy bình thường) vốn là trở ngại lớn trong phát triển phần mềm truyền thống.

Nhằm đảm bảo tính di động và khả năng tương tác (interoperability), các container hiện đại tuân thủ những tiêu chuẩn mở, điển hình là **Open Container Initiative**

(OCI) [1]. Các ví dụ phổ biến của ứng dụng container hóa là dịch vụ web chạy trong Docker container, hoặc các kiến trúc microservices được điều phối bởi Kubernetes. Trong kiến trúc đó, mỗi thành phần hoạt động độc lập trong container riêng, nhưng vẫn phối hợp nhịp nhàng để tạo nên một ứng dụng thống nhất.

1.1.2. Đặc điểm của hệ thống containerized applications

Các hệ thống ứng dụng được container hóa thể hiện một số đặc điểm then chốt, khiến chúng trở nên hữu ích vượt trội trong các kiến trúc cloud-native:

- ❖ **Tính độc lập:** Các container sử dụng các cơ chế ảo hóa ở cấp độ hệ điều hành để đảm bảo sự độc lập. Ví dụ, trên Linux, chúng sử dụng namespaces (để cô lập không gian quy trình, mạng, hệ thống tệp tin) và control groups (cgroups) (để giới hạn và quản lý tài nguyên như CPU, bộ nhớ). Điều này đảm bảo rằng các lỗi hoặc vấn đề về hiệu suất phát sinh trong một container sẽ không lan truyền sang các container khác hoặc ảnh hưởng đến toàn bộ máy chủ.
- ❖ **Tính di động:** Một khi đã được xây dựng (build), các *image* (ảnh) của container có thể được triển khai trên bất kỳ nền tảng nào tương thích với chuẩn container đó. Đặc điểm này tạo điều kiện cho việc chuyển đổi liền mạch giữa các giai đoạn phát triển (development), kiểm thử (testing), và sản xuất (production).
- ❖ **Khả năng mở rộng:** Container khởi động cực kỳ nhanh. Điều này cho phép hệ thống thực hiện mở rộng ngang (horizontal scaling) – tức là nhanh chóng khởi chạy thêm nhiều bản sao (instances) của ứng dụng – để xử lý các đợt tải tăng đột biến. Đây là yếu tố lý tưởng cho các kiến trúc microservices, nơi các dịch vụ nhỏ có thể được mở rộng độc lập.
- ❖ **Thiết kế gọn nhẹ:** Nhờ việc chia sẻ chung nhân của máy chủ và chỉ bao gồm các thư viện và tệp nhị phân thiết yếu, container tiêu thụ ít tài nguyên hơn đáng kể so với máy ảo. Điều này cho phép đạt mật độ triển khai cao hơn, nghĩa là chạy được nhiều container hơn trên cùng hạ tầng phẳng cứng so với VM.
- ❖ **Khả năng chịu lỗi và tính bất biến:** Các *image* container được thiết kế để trở thành những bản thiết kế *bất biến* (immutable). Điều này thúc đẩy tính nhất quán: nếu cần cập nhật, một container mới sẽ được tạo ra từ một *image* mới để

thay thế container cũ, thay vì vá lỗi trực tiếp trên container đang chạy. Khi kết hợp với các công cụ điều phối (như Kubernetes), hệ thống có thể tự động hóa việc xử lý lỗi (failover) và thực hiện các bản cập nhật một cách an toàn.

- ❖ Hiệu quả tài nguyên: Thông qua cgroups, việc sử dụng CPU, bộ nhớ và lưu trữ được kiểm soát chặt chẽ và hiệu quả, giúp giảm chi phí vận hành (overhead) trong các môi trường triển khai dày đặc.

Những đặc điểm này làm cho các ứng dụng container hóa trở nên đặc biệt phù hợp cho các trường hợp sử dụng như quy trình CI/CD, các tác vụ lặp lại và thúc đẩy phát triển DevOps.

1.2. Các thách thức trong điều tra số cho hệ thống containerized applications

Các phương pháp điều tra số truyền thống, vốn dựa vào phân tích dữ liệu vĩnh viễn (persistent data) trên ổ đĩa vật lý, không còn phù hợp với môi trường container hóa. Bản chất cốt lõi của container với tính biến động cao và đặc tính tạm thời cùng với các lớp trừu tượng phức tạp của chúng, tạo ra những thách thức đặc thù. Điều này đòi hỏi các nhà điều tra phải phát triển và áp dụng những phương pháp luận cũng như công cụ chuyên biệt.

1.2.1. Tính tạm thời

Đây là thách thức cốt lõi và lớn nhất [4]. Các container về bản chất được thiết kế với vòng đời ngắn; chúng thường được tạo, chạy và bị hủy bỏ tự động theo một chu trình liên tục, đặc biệt trong các môi trường auto-scaling hoặc self-healing. Vấn đề nằm ở chỗ, khi một container bị chấm dứt, toàn bộ dữ liệu runtime của nó bao gồm trạng thái bộ nhớ, nhật ký hoạt động và các tệp tin tạm thời sẽ mất vĩnh viễn nếu chúng không được cấu hình lưu trữ ở bên ngoài (chẳng hạn như persistent volumes hoặc central logging). Chính vì lẽ đó, việc thu thập bằng chứng kịp thời trở nên vô cùng quan trọng. Bản chất "thoáng qua" này có nguy cơ xóa sạch các dấu hiệu xâm phạm (IoCs) trước khi các nhà điều tra an ninh có đủ thời gian để nhận biết và thực hiện hành động cần thiết.

1.2.2. Biến động dữ liệu

Tính biến động của dữ liệu là một hệ quả trực tiếp từ bản chất tạm thời của container. Trong quá trình hoạt động, container liên tục tạo ra dữ liệu và lưu trữ chúng trong bộ nhớ (RAM) hoặc trên các phân vùng không bền vững (non-persistent storage). Toàn bộ khối dữ liệu này sẽ "bốc hơi" ngay lập tức, không thể khôi phục, khi container dừng hoạt động hoặc được khởi động lại.

Điều này đặt ra một thách thức nghiêm trọng cho công tác điều tra số. Nếu không có các công cụ chuyên dụng để kịp thời thực hiện snapshot bộ nhớ và đĩa của container đang chạy, các tạo tác điều tra (forensic artifacts) cực kỳ quan trọng sẽ vĩnh viễn không thể truy cập được. Các bằng chứng như lịch sử tiến trình (process histories), trạng thái các kết nối mạng đang hoạt động, hay thậm chí các khóa mã hóa đang lưu trong bộ nhớ, đều sẽ bị mất. Khả năng mất bằng chứng tức thời này làm phức tạp đáng kể quá trình điều tra trực tiếp (live forensics).

1.2.3. Multi-tenancy và quyền riêng tư

Trong các môi trường tài nguyên chia sẻ (shared clusters) điển hình như Kubernetes, nhiều "người thuê" (tenants) – có thể là các ứng dụng, phòng ban, hay khách hàng riêng biệt – cùng tồn tại và vận hành trên một cơ sở hạ tầng vật lý chung (máy chủ, mạng). Kiến trúc này đặt ra hai rủi ro nghiêm trọng cho quá trình điều tra.

Một là nguy cơ lây nhiễm chéo bằng chứng (evidence cross-contamination) [4]. Khi dữ liệu hoặc dấu vết hành vi của một người thuê có thể vô tình bị trộn lẫn vào dữ liệu của người thuê khác. Hai là rủi ro vi phạm quyền riêng tư [4]. Trong quá trình điều tra một người thuê, nhà điều tra rất có khả năng vô tình truy cập vào dữ liệu nhạy cảm của những người thuê khác hoàn toàn không liên quan.

Chính vì vậy, việc cô lập và trích xuất dữ liệu chính xác chỉ của một người thuê cụ thể mà không làm ảnh hưởng đến hoạt động hoặc xâm phạm quyền riêng tư của các bên khác là một thách thức kỹ thuật và pháp lý vô cùng lớn. Bài toán này càng trở nên phức tạp khi phải tuân thủ các quy định bảo vệ dữ liệu nghiêm ngặt như GDPR [2].

1.2.4. Phụ thuộc vào nhà cung cấp và quyền truy cập hạn chế

Khi các container được vận hành trên nền tảng đám mây công cộng (public cloud) hoặc thông qua các dịch vụ được quản lý (Managed Kubernetes), quyền kiểm soát cơ sở hạ tầng vật lý hoàn toàn thuộc về nhà cung cấp. Thực tế này giới hạn nghiêm ngặt quyền truy cập trực tiếp (direct access) của các nhà điều tra.

Họ thường không có khả năng truy cập vào hệ thống máy chủ (host systems) nơi container đang chạy, đồng thời cũng không thể trích xuất các metadata (siêu dữ liệu) ở tầng thấp của container. Hơn nữa, các Thỏa thuận mức độ dịch vụ (SLAs) [4] không phải lúc nào cũng đảm bảo quy trình hỗ trợ điều tra số một cách rõ ràng. Điều này có thể dẫn đến sự chậm trễ nghiêm trọng trong việc yêu cầu và nhận dữ liệu, hoặc thậm chí chỉ nhận được các bộ dữ liệu không đầy đủ.

1.2.5. Khối lượng dữ liệu lớn và quản lý nhật ký

Các hệ thống container hóa vốn tạo ra một khối lượng nhật ký (logs) khổng lồ và liên tục. Dữ liệu này được tổng hợp từ nhiều nguồn khác nhau: từ chính container runtime, từ các công cụ điều phối như Kubernetes (ví dụ, qua audit logs), và từ bên ngoài các ứng dụng. Quy mô của lượng log này có thể dễ dàng đạt đến hàng terabyte hoặc thậm chí petabyte [4].

Việc phân tích một khối lượng dữ liệu khổng lồ như vậy vốn đã là một thách thức lớn, nhất là khi dữ liệu thường bị phân mảnh và lưu trữ rác ở nhiều nơi.Thêm vào đó, nguy cơ kẻ tấn công cố ý làm xáo trộn hoặc xóa nhật ký để che giấu dấu vết khiến bài toán càng thêm phức tạp. Điều này đòi hỏi phải có các hệ thống quản lý log tập trung đủ mạnh mẽ, kết hợp với các kỹ thuật lọc và tương quan, dẫn đến việc gia tăng đáng kể yêu cầu về tài nguyên tính toán và lưu trữ phục vụ điều tra.

1.2.6. Mã hóa và kiểm soát truy cập

Để đảm bảo an toàn, dữ liệu bên trong container bao gồm cả dữ liệu "nghi" (at rest) lẫn dữ liệu "đang truyền" (in transit) và các secret (như khóa API, mật khẩu) đều được mã hóa. Việc điều tra các dữ liệu này đòi hỏi khả năng khôi phục khóa (key recovery). Quá trình này vấp phải hai trở ngại chính: một là các rào cản pháp lý liên quan đến quyền truy cập khóa; hai là thách thức kỹ thuật, đòi hỏi các công cụ phân tích runtime

phức tạp để kịp thời trích xuất khóa mã hóa từ bộ nhớ container đang chạy trước khi nó bị phá hủy [4].

1.2.7. Chain of Custody

Chain of Custody là quy trình tài liệu hóa chi tiết, liên tục về việc ai đã thu thập, xử lý và lưu trữ bằng chứng để đảm bảo tính toàn vẹn của nó. Tuy nhiên, trong các kiến trúc phân tán, nơi các container có thể được di chuyển tự động qua nhiều máy chủ, nhiều trung tâm dữ liệu, thậm chí qua các khu vực pháp lý (jurisdictions) khác nhau trên toàn cầu, việc duy trì chuỗi này trở nên vô cùng phức tạp. Theo dõi nguồn gốc (provenance) và đảm bảo một chuỗi bảo quản không bị gián đoạn trong môi trường linh động như vậy là một thách thức lớn, gây rủi ro trực tiếp cho tính hợp pháp (admissibility) của bằng chứng khi trình ra trước tòa [4].

1.2.8. Hạn chế về công cụ và kỹ thuật

Một trở ngại đáng kể là phần lớn các công cụ điều tra số truyền thống được thiết kế để làm việc với hệ thống tệp (filesystem) và hệ điều hành tiêu chuẩn. Chúng không được tối ưu hóa để phân tích hay diễn giải các cấu trúc đặc thù của container [4].

Cụ thể, các công cụ này thường gặp khó khăn lớn khi phân tích hệ thống tệp phân lớp (layered filesystems) như OverlayFS hay AUFS của Docker. Tương tự, chúng có thể không "hiểu" được các siêu dữ liệu phức tạp do các hệ thống điều phối (orchestration metadata) như Kubernetes tạo ra. Thực trạng này đòi hỏi một trong hai hướng: hoặc phải điều chỉnh (adaptations) các công cụ hiện có, hoặc phải phát triển các kỹ thuật hoàn toàn mới. Một ví dụ là xu hướng sử dụng eBPF (extended Berkeley Packet Filter) để thực hiện giám sát và thu thập bằng chứng ở cấp độ kernel trong thời gian thực (runtime monitoring), cho phép thu thập dữ liệu mà không cần can thiệp trực tiếp vào container [4].

1.3. Đề xuất quy trình thu thập và phân tích

Để giải quyết hiệu quả các thách thức đặc thù trong môi trường container, việc áp dụng một quy trình điều tra số có cấu trúc chặt chẽ là một yêu cầu bắt buộc. Quy trình được đề xuất dưới đây được xây dựng dựa trên các khung sườn tiêu chuẩn công

nghiệp, tiêu biểu là NIST (Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ) [3], đồng thời kết hợp các thực tiễn chuyên ngành.

Quy trình này được thiết kế nhằm bao quát toàn bộ các giai đoạn, từ khâu chuẩn bị, thiết lập kỹ thuật ban đầu cho đến giai đoạn khắc phục sự cố. Mục tiêu trọng tâm là đảm bảo tính toàn diện, duy trì tính toàn vẹn và bảo toàn giá trị pháp lý của bằng chứng. Nền tảng của quy trình là việc xây dựng một pipeline giám sát chủ động, từ đó mở rộng và tích hợp các kỹ thuật phân tích chuyên sâu khi sự cố xảy ra.

1.3.1. Thiết lập pipeline

Đây là giai đoạn mang tính chủ động, tập trung vào việc xây dựng "sẵn sàng điều tra" (forensic readiness) trước khi bất kỳ sự cố nào thực sự xảy ra. Bắt đầu bằng việc xây dựng một **Kế hoạch ứng phó sự cố** (Incident Response Plan - IRP) chi tiết, trong đó phân định rõ vai trò, trách nhiệm của từng thành viên và thiết lập các kênh liên lạc khẩn cấp. Song song đó, việc xác định và triển khai bộ công cụ (toolset) kỹ thuật là tối quan trọng. Bộ công cụ này cần bao gồm các giải pháp giám sát container chuyên dụng, chẳng hạn như Sysdig, để có khả năng quan sát sâu các hoạt động bên trong container, cùng với các giải pháp EDR như CrowdStrike để bảo vệ và thu thập dữ liệu từ các máy chủ (hosts).

Quan trọng không kém của giai đoạn chuẩn bị là cấu hình **lưu trữ và tổng hợp nhật ký**. Để vô hiệu hóa "tính tạm thời" của container, việc cấu hình lưu trữ bền vững cho nhật ký là bắt buộc. Kỹ thuật phổ biến là map các thư mục nhật ký quan trọng từ bên trong container ra các phân vùng lưu trữ (volumes) trên máy chủ host. Bước tiếp theo là tích hợp toàn bộ nhật ký vào một hệ thống SIEM ví dụ như AWS CloudWatch, Splunk, hay ELK Stack. Mục tiêu của hệ thống SIEM là để tổng hợp dữ liệu từ mọi nguồn, bao gồm bản thân các container, máy chủ host và cả hệ thống đi kèm như Kubernetes. Đảm bảo hệ thống tuân thủ các tiêu chuẩn bảo mật liên quan, vốn thường yêu cầu khả năng ghi nhật ký và kiểm toán nghiêm ngặt.

1.3.2. Quy trình thu thập khi xảy ra sự cố

Khi một sự cố được phát hiện, thường là thông qua việc giám sát liên tục các cảnh báo từ hệ thống SIEM hoặc các công cụ giám sát runtime, quy trình phản ứng nhanh phải được kích hoạt ngay lập tức. Một ví dụ điển hình là khi hệ thống phát hiện một sự kiện đáng ngờ (runtime event), chẳng hạn như một tiến trình shell tạo ra kết nối mạng ra bên ngoài, mang dấu hiệu của một reverse shell.

Hành động tức thời là ngăn chặn nhằm cô lập container bị ảnh hưởng, kìm hãm sự lây lan của kẻ tấn công sang các container khác hoặc máy chủ host. Tùy thuộc vào tình huống, các biện pháp có thể bao gồm tạm dừng container, ngắt kết nối mạng của nó, hoặc xóa pod (trong khi vẫn giữ lại một bản sao để điều tra).

Ngay sau đó là giai đoạn thu thập, vốn chạy đua với thời gian để thu giữ các bằng chứng dễ bay hơi trước khi chúng bị mất. Dữ liệu phải được thu thập một cách có hệ thống từ nhiều nguồn:

- **Nhật ký:** Bao gồm các bản ghi kiểm toán (audit logs) từ hệ thống điều phối như Kubernetes, nhật ký từ dịch vụ đám mây (ví dụ: AWS CloudTrail) và dữ liệu dòng chảy mạng (network flows) như VPC Flow Logs.
- **Trạng thái Container:** Cân lưu lại image gốc của container (sử dụng Docker CLI) để phục vụ phân tích tinh, thực hiện kết xuất bộ nhớ (memory dumps) của container đang chạy bằng các công cụ như Volatility (nếu có thể) và tạo một bản sao toàn vẹn của hệ thống tệp tin (filesystem).

Toàn bộ quá trình trích xuất metadata và các artifacts này phải được thực hiện thông qua các công cụ điều tra số chuyên dụng. Việc này đảm bảo bằng chứng gốc không bị thay đổi, qua đó duy trì tính toàn vẹn của Chain of Custody.

1.3.3. Cách thức phân tích các loại bằng chứng

Sau khi được thu thập, toàn bộ dữ liệu sẽ được đưa vào một môi trường phân tích an toàn và cô lập. Quá trình này bắt đầu bằng việc phân tích đa nguồn, trong đó các nhà điều tra sẽ kiểm tra chi tiết nhật ký, hệ thống tệp (tập trung vào các tệp lạ hoặc bị thay đổi) và dữ liệu lưu lượng mạng để xác định các IoCs. Ví dụ, việc phân tích tệp chuyên sâu là cần thiết để xác định sự hiện diện của mã độc.

Bước quan trọng tiếp theo là tương quan dữ liệu, tức là tích hợp thông tin từ các nguồn khác nhau để dựng nên bức tranh toàn cảnh. Chẳng hạn, các công cụ như Autopsy hoặc Oxygen Forensic có thể được dùng để đối chiếu một sự kiện từ Kubernetes (như "một pod mới được tạo") với dữ liệu dòng chảy VPC, qua đó phát hiện ra rằng pod đó đang kết nối đến một địa chỉ IP độc hại đã biết.

Một thách thức riêng biệt là xử lý các bằng chứng bị mã hóa. Đối với loại dữ liệu này, nhà điều tra cần áp dụng các kỹ thuật để thu hồi encryption key, ví dụ như phân tích các bản kết xuất bộ nhớ hoặc phải thông qua các yêu cầu pháp lý chính thức đến nhà cung cấp dịch vụ hoặc quản trị viên hệ thống.

1.3.4. Xây dựng timeline

Mục tiêu cốt lõi của giai đoạn này là tái tạo lại một cách chính xác chuỗi các sự kiện đã diễn ra. Điều này đòi hỏi việc xây dựng một mốc thời gian (timeline) chi tiết bằng cách sắp xếp và đối chiếu tuần tự các dấu thời gian (timestamps) thu thập được từ tất cả các nguồn: nhật ký, các sự kiện từ hệ thống điền phôi và các dòng chảy mạng.

Thông qua việc này, nhà điều tra có thể tái lập lại toàn bộ trình tự tấn công. Ví dụ, một timeline hoàn chỉnh có thể cho thấy vào (10:00 AM), kẻ tấn công có được truy cập ban đầu (initial access) vào một container dịch vụ web thông qua thông tin đăng nhập yếu, chỉ 5 phút sau (10:05 AM), chúng thực hiện leo thang đặc quyền (escalation) bằng cách khai thác Docker socket bị cấu hình sai và đến (10:15 AM), chúng bắt đầu trích xuất dữ liệu (exfiltration) ra bên ngoài.

Để đảm bảo tính chính xác, các frameworks chuyên dụng như TamForen (một framework điều tra số cho container) có thể được sử dụng. Các công cụ này giúp đảm bảo tính toàn vẹn của dữ liệu và hỗ trợ xác định các cột mốc sự kiện quan trọng.

1.3.5. Điều tra nguyên nhân / vị trí sự cố

Giai đoạn này đi sâu vào phân tích để trả lời các câu hỏi mấu chốt: "Tại sao?" và "Bằng cách nào?" sự cố có thể xảy ra. Trọng tâm chính là việc tái lập lại toàn bộ đường đi của cuộc tấn công. Quá trình này bắt đầu từ việc xác định điểm xâm nhập

ban đầu, chẳng hạn như một dịch vụ chứa lỗ hổng RCE bị khai thác. Tiếp theo, nhà điều tra phải làm rõ cách thức kẻ tấn công leo thang đặc quyền, ví dụ như lợi dụng một "privileged container" (container có đặc quyền cao). Sau đó, cần phân tích các kỹ thuật duy trì truy cập (persistence), như việc tạo ra các "backdoor pods" (pod cửa hậu) hoặc cài cắm cronjob. Cuối cùng, phải điều tra quá trình di chuyển ngang (lateral movement) để hiểu rõ kẻ tấn công đã mở rộng phạm vi kiểm soát từ container ban đầu sang các container khác trong cùng mạng nội bộ như thế nào.

Song song với việc truy vết, việc đánh giá tác động toàn diện của vụ tấn công là bắt buộc, bao gồm mức độ lộ lọt dữ liệu (data exposure) và phạm vi các hệ thống khác bị ảnh hưởng. Mục tiêu cuối cùng là xác định chính xác root cause đã tạo điều kiện cho cuộc tấn công, ví dụ như một lỗi cấu hình sai trong RBAC (Role-Based Access Control) của Kubernetes, cho phép container truy cập trái phép vào API server.

1.3.6. Quy trình khắc phục

Giai đoạn cuối cùng tập trung vào việc phục hồi hệ thống và cải tiến quy trình. Công tác này bắt đầu bằng việc khắc phục, tức là loại bỏ triệt để nguyên nhân gốc rễ đã được xác định: gỡ bỏ mã độc, và các lỗ hổng phần mềm và sửa chữa các cấu hình sai. Đồng thời, các biện pháp tăng cường bảo mật phải được áp dụng ngay lập tức, như thực thi nguyên tắc đặc quyền tối thiểu (least privilege) hoặc mã hóa dữ liệu nhạy cảm, để ngăn chặn sự cố tái diễn.

Tiếp theo là giai đoạn phục hồi (Recovery), đưa hệ thống trở lại trạng thái hoạt động bình thường và an toàn. Trước khi quay lại môi trường sản xuất (production), hệ thống bắt buộc phải trải qua các quy trình kiểm thử kỹ lưỡng nhằm xác nhận tính ổn định và bảo mật.

Bước then chốt khép lại toàn bộ quy trình là rút kinh nghiệm. Toàn bộ sự cố bao gồm những gì đã xảy ra, cách thức đội ngũ đã phản ứng và kết quả cuối cùng phải được tài liệu hóa (document) một cách chi tiết. Những phân tích này là cơ sở quan trọng nhất để cập nhật và cải tiến IRP. Chẳng hạn, nếu quá trình điều tra cho thấy việc

phát hiện ban đầu quá chậm, đội ngũ có thể quyết định bổ sung các quy tắc tự động hóa để nhận diện các hành vi tương tự nhanh hơn trong tương lai.

Bảng 1. Tổng hợp các giai đoạn theo quy trình để xuất

Giai đoạn	Hoạt động Chính	Công cụ	Mục tiêu
Thiết lập pipeline	Cấu hình logs, monitoring	Sysdig, SIEM	Sẵn sàng forensic
Thu thập khi sự cố	Cô lập, capture data	Docker CLI, Volatility	Bảo tồn bằng chứng
Phân tích bằng chứng	Xét IoCs, correlate data	Autopsy, eBPF	Hiểu hành vi tấn công
Xây dựng timeline	Sắp xếp events chronologically	TamForen, log analyzers	Tái tạo chuỗi sự kiện
Điều tra nguyên nhân	Xác định access, escalation	Network tools, forensics kits	Tìm lỗ hổng cốt lõi
Khắc phục	Xóa threats, khôi phục	Patch tools, RBAC configs	Ngăn tái diễn

CHƯƠNG 2: XÂY DỰNG PIPELINE TỰ ĐỘNG HÓA VIỆC THU THẬP CÁC ARTIFACTS TỪ ORCHESTRATION

2.1. Xác định các nguồn artifacts quan trọng

Các hệ thống containerized applications quy mô lớn đòi hỏi cơ chế điều phối tự động và quản lý tập trung để đảm bảo vận hành hiệu quả. Kubernetes hiện đã trở thành tiêu chuẩn de facto cho nhiệm vụ này, giữ vai trò trung tâm trong các quy trình DevOps. Bằng cách hiện thực hóa nguyên lý tự động hóa hạ tầng, Kubernetes tích hợp trực tiếp với các công cụ CI/CD (như Jenkins, GitLab CI hay ArgoCD) để tự động hóa toàn bộ vòng đời ứng dụng.

Về mặt kiến trúc, một môi trường Kubernetes (cluster) bao gồm các master node chịu trách nhiệm điều phối và các worker node nơi ứng dụng thực sự chạy bên trong các pod. Các thành phần khác như service, deployment, và ingress chịu trách nhiệm định tuyến, kiểm soát cập nhật và quản lý truy cập từ bên ngoài.

Từ góc độ điều tra số, kiến trúc đa tầng và phân tán này tạo ra một hệ thống artifacts tương ứng phức tạp, trong đó nhật ký (logs) là nguồn bằng chứng then chốt. Để tái dựng lại chuỗi sự kiện của một sự cố an ninh, việc thu thập và phân tích tương quan giữa các nguồn log này mang tính sống còn. Các nguồn logs này có thể được phân loại một cách rõ ràng theo ba cấp độ của hệ thống: cấp độ ứng dụng, cấp độ hạ tầng (node) và cấp độ điều khiển (control plane).

Log ứng dụng, hay log container, bao gồm các luồng dữ liệu stdout và stderr do chính các tiến trình bên trong container tạo ra. Kubelet có nhiệm vụ thu thập các luồng này và lưu trữ chúng trên hệ thống tệp của worker node. Về mặt điều tra, các log này có giá trị cao trong việc xác định điểm xâm nhập ban đầu (initial foothold). Việc phân tích chúng có thể tiết lộ bằng chứng về việc khai thác lỗ hổng (ví dụ: các chuỗi SQL injection, lối tràn bộ đệm) hoặc các lệnh được kẻ tấn công thực thi ngay sau khi chiếm được quyền truy cập. Tuy nhiên, bản chất tạm thời (ephemeral) của Pods là một thách thức lớn; nếu một Pod bị xóa, các log liên quan trên node cũng sẽ bị hủy. Điều này

khiến việc triển khai một hệ thống thu thập log tập trung trở thành yêu cầu bắt buộc để đảm bảo tính toàn vẹn của bằng chứng.

Ở cấp độ hạ tầng, **log của node** (host) cung cấp bối cảnh về các sự kiện xảy ra trên máy chủ vật lý hoặc máy ảo. Các nguồn log chính tại đây bao gồm log hệ điều hành (như syslog, journald), log của container runtime (như containerd, dockerd), và log của kubelet. Giá trị pháp lý của các log này nằm ở khả năng phát hiện các nỗ lực leo thang đặc quyền ra khỏi container (container escape) hoặc các truy cập trái phép trực tiếp vào worker node. Ví dụ, journald có thể ghi lại các phiên SSH đáng ngờ vào node, trong khi log của container runtime có thể chứa dấu vết của các lệnh gọi API cấp thấp bất thường nhằm lợi dụng lỗ hổng của runtime.

Nguồn log quan trọng và có giá trị nhất cho điều tra số trong môi trường Kubernetes là **log của control plane**, đặc biệt là Kubernetes Audit Logs. Mặc dù log của API server ghi lại mọi yêu cầu, Audit Logs được thiết kế chuyên biệt cho mục đích an ninh, cung cấp một bản ghi chi tiết về ai (user/service account), đã làm gì (verb), với tài nguyên nào (resource), và khi nào. Đây là nguồn bằng chứng chính để tái tạo lại các hành động của kẻ tấn công ở cấp độ cụm (cluster), chẳng hạn như việc thực thi lệnh từ xa vào một Pod (pods/exec), truy cập các dữ liệu nhạy cảm (secrets/get), tạo ra các tài nguyên mới (ví dụ: deployment/create cho mục đích đào tiền ảo), hoặc các nỗ lực leo thang đặc quyền trong cụm (clusterrolebinding/create).

Về cơ bản, không một loại log đơn lẻ nào là đủ. Giá trị điều tra số chỉ có thể đạt được thông qua việc xây dựng một pipeline tự động hóa việc tổng hợp, chuẩn hóa và phân tích tương quan tất cả các nguồn log nói trên. Đây là cách duy nhất để xây dựng được một bức tranh toàn cảnh và chính xác về sự cố an ninh đã xảy ra.

2.2. Triển khai hệ thống Kubernetes với MicroK8s

Để mô phỏng hoạt động của một hệ thống containerized applications do Kubernetes điều phối, nhóm đã xây dựng một mô hình cluster đa node bằng MicroK8s. Môi trường này được thiết kế chuyên biệt để sẵn sàng cho việc điều tra số (forensic-ready), trong đó triển khai ứng dụng web có lỗ hổng (Juice Shop) làm mục tiêu. Về mặt hạ

tầng, hệ thống bao gồm ba máy ảo chạy Ubuntu Server 24, được phân chia thành một node "master" và hai node "worker".

Instances (5) Info								
		Connect		Instance state ▼		Actions ▼		Launch instances
				All states ▼				
	Name 🔗	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	
<input type="checkbox"/>	Spunk indexer	i-02de201e5233c46af	Running Q Q	t2.large	2/2 checks passed	View alarms +	ap-southeast-1a	
<input checked="" type="checkbox"/>	Master node	i-00824e24978ed6270	Running Q Q	t3.large	3/3 checks passed	View alarms +	ap-southeast-1b	
<input type="checkbox"/>	Splunk_search_head	i-0e473a2b6b415eba1	Running Q Q	t2.large	2/2 checks passed	View alarms +	ap-southeast-1b	
<input checked="" type="checkbox"/>	Worker2 node	i-077bbe1e59ed2d847	Running Q Q	t2.medium	2/2 checks passed	View alarms +	ap-southeast-1b	
<input checked="" type="checkbox"/>	Worker1 node	i-085e3569983a2dc08	Running Q Q	t2.medium	2/2 checks passed	View alarms +	ap-southeast-1b	

Quá trình thiết lập bắt đầu trên master node, nơi MicroK8s được cài đặt và các addons thiết yếu như dns, registry, ingress, metrics-server, và dashboard đã được kích hoạt. Sau khi master node thực thi lệnh microk8s add-node để tạo chuỗi lệnh tham gia (join command), các worker node đã sử dụng lệnh microk8s join ... --worker này để kết nối vào cluster. Quá trình gia nhập đã diễn ra thành công, các node worker được tự động cấu hình và cluster đi vào trạng thái sẵn sàng.

```
ubuntu@ip-172-31-26-89:~$ microk8s enable dns registry ingress metrics-server dashboard
Infer repository core for addon dns
Infer repository core for addon registry
Infer repository core for addon ingress
Infer repository core for addon metrics-server
Infer repository core for addon dashboard
WARNING: Do not enable or disable multiple addons in one command.
This form of chained operations on addons will be DEPRECATED in the future.
Please, enable one addon at a time: 'microk8s enable <addon>'
```

```
worker1
ubuntu@ip-172-31-18-131:~$ sudo usermod -a -G microk8s $USER
ubuntu@ip-172-31-18-131:~$ newgrp microk8s
ubuntu@ip-172-31-18-131:~$ microk8s join 172.31.26.89:25000/b9bf46b8f511283f98be93375667a965/b286804462de --worker
Contacting cluster at 172.31.26.89

The node has joined the cluster and will appear in the nodes list in a few seconds.

This worker node gets automatically configured with the API server endpoints.
If the API servers are behind a loadbalancer please set the '--refresh-interval' to '0s' in:
  /var/snap/microk8s/current/args/apiserver-proxy
and replace the API server endpoints with the one provided by the loadbalancer in:
  /var/snap/microk8s/current/args/traefik/provider.yaml

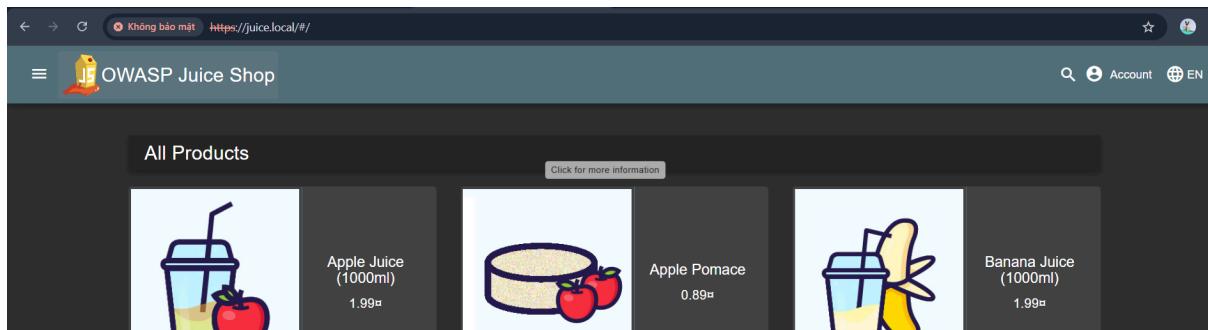
Successfully joined the cluster.
ubuntu@ip-172-31-18-131:~$
```

Giai đoạn thiết lập hoàn tất với việc triển khai ứng dụng mục tiêu OWASP Juice Shop lên cluster bằng Helm. Một tài nguyên Ingress cũng được định nghĩa (qua tệp juice-shop-ingress.yaml) để ánh xạ ứng dụng với tên miền juice.local. Việc truy

cập thành công ứng dụng Juice Shop tại <https://juice.local> qua trình duyệt đã xác nhận toàn bộ quá trình cài đặt hoàn tất.

```
ubuntu@ip-172-31-26-89:~$ microk8s helm3 install juice-shop oci://ghcr.io/securecodebox/helm/juice-shop \
--namespace default \
--set service.type=NodePort
Pulled: ghcr.io/securecodebox/helm/juice-shop:5.1.0
Digest: sha256:1f42f901fdb9fd8959292b5fb135c37bc9ac478627db1f72a0f28ea59a2a9856
NAME: juice-shop
LAST DEPLOYED: Wed Oct 22 19:09:43 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath=".spec.ports[0].nodePort" services juice-shop)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath=".items[0].status.addresses[0].address")
  echo http://$NODE_IP:$NODE_PORT
```

```
root@ip-172-31-26-89:/home ~ + | juice-shop-ingress.yaml
GNU nano 7.2
apiVersion: v1
kind: Service
metadata:
  name: juice-shop
  namespace: default
spec:
  selector:
    app: juice-shop
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: ClusterIP
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: juice-shop
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/proxy-body-size: "50m"
    nginx.ingress.kubernetes.io/enable-access-log: "true"
spec:
  rules:
    - host: juice.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: juice-shop
                port:
                  number: 80
```

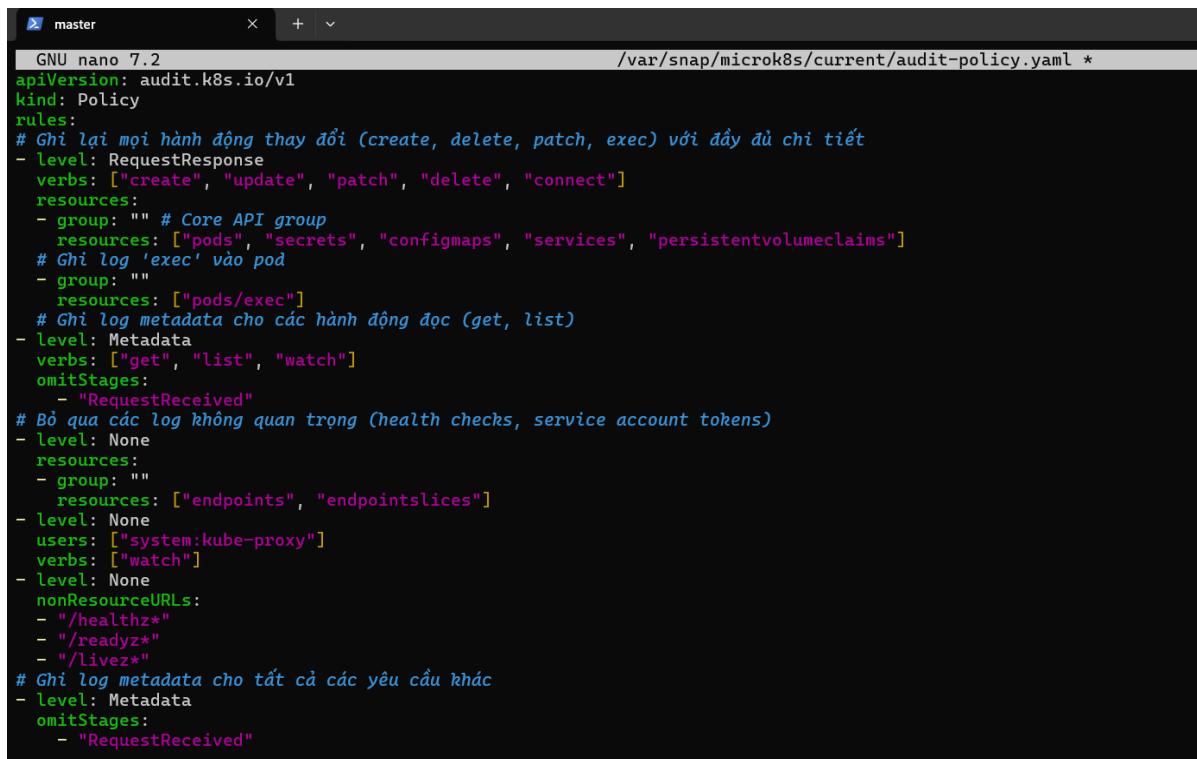


2.3. Cấu hình việc ghi logs của MicroK8s

Để thu thập bằng chứng ở cấp độ điều phối (orchestration), hai lớp ghi log đã được cấu hình: Kubernetes Audit Logging và Host-level Auditing (sử dụng auditd). Đối với Kubernetes, tính năng audit log đã được kích hoạt bằng cách sửa đổi trực tiếp tệp cấu hình của kube-apiserver (tại /var/snap/microk8s/current/args/kube-apiserver). Các tham số chính đã được bổ sung để xác định đường dẫn tệp log (--audit-log-path), tệp chính sách (--audit-policy-file), cùng các quy tắc xoay vòng log (log rotation) như maxage, maxbackup, và maxsize.

```
# Disable the watchlist feature until k8s-dqlite can handle it
--feature-gates=WatchList=false

--audit-log-path=/var/snap/microk8s/current/audit.log
--audit-log-maxage=30
--audit-log-maxbackup=10
--audit-log-maxsize=100
--audit-policy-file=/var/snap/microk8s/current/audit-policy.yaml
```



The screenshot shows a terminal window titled "master" with the command "/var/snap/microk8s/current/audit-policy.yaml *". The file content is displayed in a nano text editor:

```
GNU nano 7.2
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
# Ghi lại mọi hành động thay đổi (create, delete, patch, exec) với đầy đủ chi tiết
- level: RequestResponse
  verbs: ["create", "update", "patch", "delete", "connect"]
  resources:
    - group: "" # Core API group
      resources: ["pods", "secrets", "configmaps", "services", "persistentvolumeclaims"]
    # Ghi log 'exec' vào pod
    - group: ""
      resources: ["pods/exec"]
    # Ghi log metadata cho các hành động đọc (get, list)
    - level: Metadata
      verbs: ["get", "list", "watch"]
      omitStages:
        - "RequestReceived"
    # Bỏ qua các log không quan trọng (health checks, service account tokens)
    - level: None
      resources:
        - group: ""
          resources: ["endpoints", "endpointslices"]
    - level: None
      users: ["system:kube-proxy"]
      verbs: ["watch"]
    - level: None
      nonResourceURLs:
        - "/healthz*"
        - "/readyz*"
        - "/livez*"
    # Ghi log metadata cho tất cả các yêu cầu khác
    - level: Metadata
      omitStages:
        - "RequestReceived"
```

Chính sách audit được thiết kế đặc biệt để ghi lại chi tiết ở mức RequestResponse (bao gồm cả nội dung request và response) cho các hành động nhạy cảm như create, update, patch, delete, và pods/exec. Trong khi đó, các hành động chỉ đọc (ví dụ: get, list) được ghi ở mức Metadata, và các sự kiện không quan trọng (như health checks) bị bỏ qua hoàn toàn (level: None) để tránh nhiều dữ liệu.

```
master x + v
ubuntu@ip-172-31-26-89:~$ sudo apt-get update
sudo apt-get install auditd -y
sudo systemctl enable auditd
sudo systemctl start auditd
Hit:1 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Hit:3 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Fetched 126 kB in 1s (202 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Song song đó, để giám sát các hoạt động ở cấp độ hệ điều hành trên từng node (Host-level Auditing), dịch vụ auditd đã được cài đặt và kích hoạt trên cả master node lẫn các worker node. Một bộ rules tùy chỉnh đã được định nghĩa trong tệp /etc/audit/rules.d/99-k8s-forensics.rules. Các luật này được thiết kế để theo dõi các hoạt động có rủi ro bảo mật cao, bao gồm:

- ❖ Mọi truy cập vào socket của Containerd (containerd.sock).
- ❖ Hoạt động thực thi các tệp nhị phân của MicroK8s (như microk8s.ctr, microk8s.kubectl).
- ❖ Bất kỳ thay đổi nào đối với tệp cấu hình Kubelet.
- ❖ Các hành vi sửa đổi tệp hệ thống nhạy cảm (ví dụ: /etc/passwd, /etc/shadow).
- ❖ Toàn bộ các lệnh được thực thi thông qua syscall execve.

Sau khi định nghĩa, các luật này đã được nạp vào auditd (qua auditctl -R) và được kiểm tra lại (bằng auditctl -l) để xác nhận chúng đã được áp dụng thành công trên tất cả các node trong cluster.

```
master x + v
ubuntu@ip-172-31-26-89:~$ sudo nano /etc/audit/rules.d/99-k8s-forensics.rules
ubuntu@ip-172-31-26-89:~$ sudo nano /etc/audit/rules.d/99-k8s-forensics.rules
ubuntu@ip-172-31-26-89:~$ sudo auditctl -R /etc/audit/rules.d/99-k8s-forensics.rules
ubuntu@ip-172-31-26-89:~$ sudo auditctl -l
-w /var/snap/microk8s/common/run/containerd.sock -p rwxa -k k8s_containerd_sock
-w /snap/bin/microk8s.ctr -p x -k k8s_crictl_exec
-w /snap/bin/microk8s.kubectl -p x -k k8s_kubectl_exec
-w /var/snap/microk8s/current/args/kubelet -p wa -k k8s_kubelet_config
-w /etc/passwd -p wa -k system_passwd_changed
-w /etc/shadow -p wa -k system_shadow_changed
-w /etc/sudoers -p wa -k system_sudoers_changed
-a always,exit -F arch=b64 -S execve -F key=command_execution
-a always,exit -F arch=b32 -S execve -F key=command_execution
```

2.4. Triển khai Fluent Bit và Suricata để thu thập và gửi logs

Để tự động hóa việc thu thập artifacts, nhóm đã thiết lập một pipeline thu thập logs tập trung. Pipeline này sử dụng Fluent Bit, được triển khai dưới dạng DaemonSet trên cụm K8s, với nhiệm vụ chuyển tiếp (forward) logs về một máy chủ trung tâm (ip-172-31-33-241). Trên máy chủ trung tâm này, Fluent Bit cũng được cài đặt và cấu hình như một dịch vụ, thực hiện việc lắng nghe (input forward) trên cổng 24224 để tiếp nhận dữ liệu log gửi về từ các node K8s.

```
root@ip-172-31-33-241:/home/ubuntu# sudo systemctl start fluent-bit
root@ip-172-31-33-241:/home/ubuntu# sudo systemctl status fluent-bit
● fluent-bit.service - Fluent Bit
   Loaded: loaded (/usr/lib/systemd/system/fluent-bit.service; enabled; preset: enabled)
     Active: active (running) since Thu 2025-10-23 18:02:30 UTC; 442ms ago
       Docs: https://docs.fluentbit.io/manual/
   Main PID: 394372 (fluent-bit)
     Tasks: 3 (limit: 9498)
    Memory: 2.9M (peak: 3.0M)
      CPU: 11ms
     CGroup: /system.slice/fluent-bit.service
             └─394372 /opt/fluent-bit/bin/fluent-bit -c /etc/fluent-bit/fluent-bit.conf

Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.9723282556] [ info] [storage] ver=1.5.3, type=memory, sync=normal, checksum=off, n
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972399681] [ info] [simd ] SSE2
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972411019] [ info] [cmetrics] version=1.0.5
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972421656] [ info] [ctraces ] version=0.6.6
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972505643] [ info] [input:forward:forward.0] initializing
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972526644] [ info] [input:forward:forward.0] storage_strategy='memory' (memory on
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972631631] [ info] [input:forward:forward.0] Listening on 0.0.0.0:24224
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972715621] [error] [output:file:file.0] unknown format json_lines. abort.
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972736268] [error] [output] failed to initialize 'file' plugin
Oct 23 18:02:30 ip-172-31-33-241 fluent-bit[394372]: [2025/10/23 18:02:30.972761823] [error] [engine] output initialization failed
root@ip-172-31-33-241:/home/ubuntu# clear
```

Cấu hình cho các agent Fluent Bit trên node K8s được quản lý thông qua ConfigMap, bao gồm các thành phần chính:

❖ [INPUT]:

- Log container: Path /var/log/containers/*.log (Tag: kube.*).
- Log hệ thống: Path /var/log/auth.log* (Tag: host.auth).
- Log auditd: Path /var/log/audit/audit.log* (Tag: host.audit).

❖ [FILTER]:

- Bộ lọc kubernetes được áp dụng cho tag kube.* để tự động làm giàu log container với các metadata của K8s (như tên pod, namespace, v.v.).
- Bộ lọc modify được áp dụng cho tất cả log (*) để thêm trường node_name, lấy giá trị từ biến môi trường NODE_NAME. Biến này được truyền vào pod thông qua fieldRef: spec.nodeName.

❖ [OUTPUT]:

- Toàn bộ log (Match *) được chuyển tiếp (forward) đến máy chủ log trung tâm tại địa chỉ Host 172.31.33.241 và Port 24224.

```

fluent-bit.conf: |
  [SERVICE]
    Flush          5
    Daemon        Off
    Log_Level     info
    Parsers_File  parsers.conf
    Environment   On
    storage.path  /fluent-bit/storage
    storage.sync   normal
    storage.checksum off
    storage.backlog.mem_limit 256MB

  [INPUT]
    Name          tail
    Tag           kube./*
    Path          /var/log/containers/*.log
    DB            /host-local-logs/flb_kube.db
    Mem_Buf_Limit 10MB
    Skip_Long_Lines On
    Refresh_Interval 10
    Multiline.Parser docker, cri
    storage.type   filesystem

  [INPUT]
    Name          tail
    Tag           host.audit
    Path          /var/log/audit/audit.log*
    DB            /host-local-logs/flb_audit.db
    Mem_Buf_Limit 5MB
    storage.type   filesystem

  [INPUT]
    Name          tail
    Tag           host.auth
    Path          /var/log/auth.log*
    DB            /host-local-logs/flb_host_auth.db
    Mem_Buf_Limit 5MB
    storage.type   filesystem

  [FILTER]
    Name      kubernetes
    Match    kube./*
    Kube_URL https://kubernetes.default.svc:443
    Merge_Log On
    Kube_Tag_Prefix kube.

  [FILTER]
    Name      modify
    Match    *
    Add      node_name ${NODE_NAME}

  [OUTPUT]
    Name      forward
    Match    *
    Host     172.31.33.241
    Port     24224
    Retry_Limit 5
    Require_ack_response False
    storage.total_limit_size 1G
    net.keepalive on
    net.connect_timeout 10
    Workers     2
  parsers.conf: |
    [PARSER]
      Name      docker
      Format   json
      Time_Key time
      Time_Format %Y-%m-%dT%H:%M:%S.%L%z

    [PARSER]
      Name      cri
      Format   regex
      Regex   ^(?:<time>[^ ]+) (?<stream>stdout|stderr) (?<flags>F|P) (?<log>.*)>
      Time_Key time
      Time_Format %Y-%m-%dT%H:%M:%S.%L%z

```

Cáu hình của DaemonSet cũng định nghĩa các volumeMounts quan trọng để cho phép Fluent Bit truy cập vào tệp cấu hình (từ ConfigMap) và các thư mục log trên máy chủ host (ví dụ: /var/log ở chế độ readOnly: true).

Quá trình truyền dữ liệu (in-flight data) giữa các agent Fluent Bit (từ node K8s) và máy chủ trung tâm được thực hiện qua giao thức Fluentd Forward Protocol. Cụ thể, plugin "forward" trong Fluent Bit sử dụng MessagePack để tuần tự hóa (serialize) các bản ghi log. Việc lựa chọn MessagePack, một định dạng nhị phân (binary format), là một quyết định có chủ đích nhằm tối ưu hiệu suất, giúp tăng tốc độ xử lý và giảm đáng kể kích thước dữ liệu truyền đi. Sau khi máy chủ trung tâm tiếp nhận và giải tuần tự hóa (deserialize) dữ liệu MessagePack, các bản ghi sẽ được lưu trữ dưới định dạng JSON tiêu chuẩn, tạo thuận lợi cho việc phân tích về sau.

```
GNU nano 7.2                                     /etc/fluent-bit/fluent-bit.conf *
#####
[SERVICE]
  Flush      5
  Daemon    Off
  Log_Level  info
  Parsers_File parsers.conf
  storage.path /var/lib/fluent-bit
  storage.sync normal
  storage.checksum off
  storage.backlog.mem_limit 256MB

#####
# ● INPUT
#####

[INPUT]
  Name      forward
  Listen    0.0.0.0
  Port      24224
  Tag       remote./*
  storage.type filesystem

#####
# ✘ FILTER
#####

[FILTER]
  Name      record_modifier
  Match    *
  Record   receiver_host ${HOSTNAME}
```

Toàn bộ pipeline đã được xác minh end-to-end. Kết nối từ master node đến cổng 24224 của máy chủ log trung tâm được kiểm tra thành công bằng nc. Ngay sau đó, một thông điệp đánh dấu (ví dụ: AUDITTEST...) đã được ghi thủ công vào tệp /var/log/audit/audit.log trên master node.

Kết quả kiểm tra trên máy chủ log trung tâm, sử dụng grep, đã tìm thấy chính xác các thông điệp đánh dấu này trong tệp log tập trung. Thử nghiệm này chứng minh rằng pipeline Fluent Bit đang hoạt động chính xác, có khả năng thu thập log từ auditd, tự động làm giàu log bằng các metadata (như node_name), và chuyển tiếp chúng về máy chủ trung tâm một cách toàn vẹn.

```

root@ip-172-31-26-89:/home/ubuntu# nc -zv 172.31.33.241 24224
Connection to 172.31.33.241 24224 port [tcp/*] succeeded!
root@ip-172-31-26-89:/home/ubuntu# MARKER="AUDITTEST-$(date +%s)"
sudo bash -c "echo \"type=TEST msg=$MARKER\" >> /var/log/audit/audit.log"
root@ip-172-31-26-89:/home/ubuntu# MARKER="AUDITTEST-$(date +%s)"
sudo bash -c "echo \"type=TEST msg=$MARKER\" >> /var/log/audit/audit.log"
root@ip-172-31-26-89:/home/ubuntu# MARKER="AUDITTEST-$(date +%s)"
sudo bash -c "echo \"type=TEST msg=$MARKER\" >> /var/log/audit/audit.log"
root@ip-172-31-26-89:/home/ubuntu# MARKER="AUDITTEST-$(date +%s)"
sudo bash -c "echo \"type=TEST msg=$MARKER\" >> /var/log/audit/audit.log"
root@ip-172-31-26-89:/home/ubuntu# MARKER="AUDITTEST-$(date +%s)"
sudo bash -c "echo \"type=TEST msg=$MARKER\" >> /var/log/audit/audit.log"

```

```

ubuntu@ip-172-31-33-241:~$ sudo grep AUDITTEST /var/log/fluent-bit-central/logs.json
{"log":"2025-10-24T07:57:17.229940+00:00 ip-172-31-26-89 sudo:      root : TTY pts/1 ; PWD=/home/ubuntu ; USER=root ; COMMAND=/usr/bin/bash
EST msg=AUDITTEST-1761292637\" >> /var/log/audit/audit.log\"", "node_name": "ip-172-31-26-89"}
{"log":"2025-10-24T07:57:19.529569+00:00 ip-172-31-26-89 sudo:      root : TTY pts/1 ; PWD=/home/ubuntu ; USER=root ; COMMAND=/usr/bin/bash
EST msg=AUDITTEST-1761292639\" >> /var/log/audit/audit.log\"", "node_name": "ip-172-31-26-89"}
{"log": "type=TEST msg=AUDITTEST-1761292637", "node_name": "ip-172-31-26-89"}
{"log": "type=TEST msg=AUDITTEST-1761292639", "node_name": "ip-172-31-26-89"}
{"log": "type=TEST msg=AUDITTEST-1761292641", "node_name": "ip-172-31-26-89"}
{"log": "type=TEST msg=AUDITTEST-1761292642", "node_name": "ip-172-31-26-89"}
{"log": "type=TEST msg=AUDITTEST-1761292643", "node_name": "ip-172-31-26-89"}
{"log": "2025-10-24T07:57:21.842741+00:00 ip-172-31-26-89 sudo:      root : TTY pts/1 ; PWD=/home/ubuntu ; USER=root ; COMMAND=/usr/bin/bash
EST msg=AUDITTEST-1761292641\" >> /var/log/audit/audit.log\"", "node_name": "ip-172-31-26-89"}
{"log": "2025-10-24T07:57:22.900357+00:00 ip-172-31-26-89 sudo:      root : TTY pts/1 ; PWD=/home/ubuntu ; USER=root ; COMMAND=/usr/bin/bash
EST msg=AUDITTEST-1761292642\" >> /var/log/audit/audit.log\"", "node_name": "ip-172-31-26-89"}
{"log": "2025-10-24T07:57:23.986047+00:00 ip-172-31-26-89 sudo:      root : TTY pts/1 ; PWD=/home/ubuntu ; USER=root ; COMMAND=/usr/bin/bash
EST msg=AUDITTEST-1761292643\" >> /var/log/audit/audit.log\"", "node_name": "ip-172-31-26-89"}

```

Để tăng cường hiệu quả cho pipeline, ngoài việc thu thập log từ host và K8s, nhóm đã triển khai Suricata để bổ sung khả năng giám sát và phân tích lưu lượng mạng, tập trung vào lưu lượng HTTP đi vào các web server. Thông qua việc phân tích sâu các gói tin dựa trên bộ quy tắc (ruleset) định sẵn, Suricata có khả năng phát hiện các dấu hiệu tấn công, mã độc và những hành vi bất thường.

Toàn bộ cảnh báo và sự kiện mạng (ví dụ: log HTTP) do Suricata phát hiện sẽ được ghi tập trung tại thư mục /var/log/suricata trên master node. Để tích hợp vào pipeline chung, thư mục này được cấu hình làm đầu vào cho Splunk Universal Forwarder, đảm bảo log được tự động thu thập và đẩy về Splunk indexer. Điểm mấu chốt của việc tích hợp này là cung cấp cho các nhà điều tra khả năng tương quan hóa sự kiện, cho phép họ liên kết một sự kiện mạng (như một yêu cầu HTTP độc hại) với các sự kiện ở cấp độ host (từ auditd) và cấp độ điều phối (từ K8s Audit Logs).

```

app-layer:
  # error-policy: ignore
  protocols:
    http:
      enabled: yes
      libhttp:
        default-config:
          personality: IDS
          request-body-limit: 0 # Set to 0 for unlimited or adjust as needed
          response-body-limit: 0 # Set to 0 for unlimited or adjust as needed

```

```

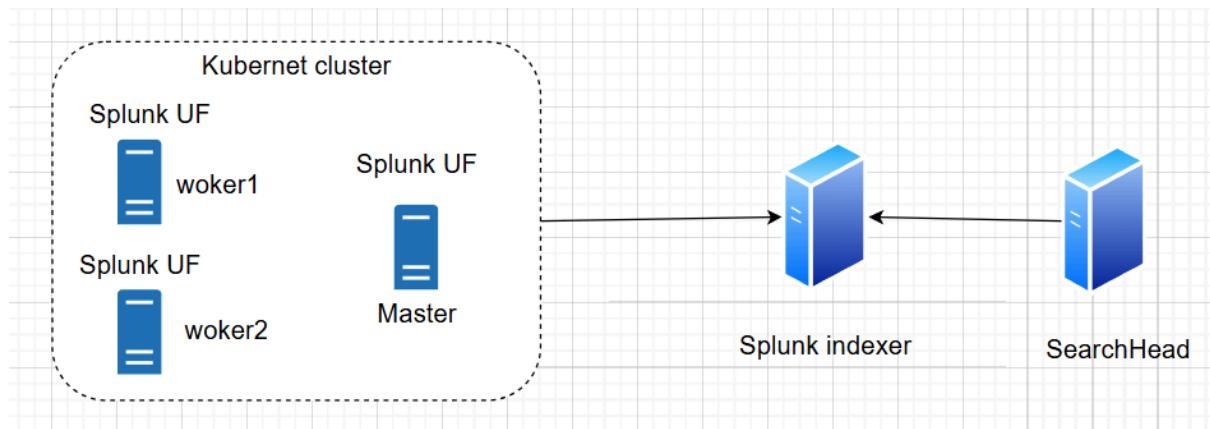
types:
  - alert:
      payload: yes          # Dump payload (Base64)
      payload-printable: yes # Dump payload ở dạng readable text (mất dữ liệu nhị phân)
      payload-buffer-size: 5mb # Kích thước tối đa mỗi payload log
      payload-length: yes   # Ghi độ dài payload
      packet: yes           # Ghi packet raw (không cần thiết nếu chỉ cần payload)
      metadata: yes          # Bao gồm metadata lớp ứng dụng
      http-body: yes         # Dump HTTP body (Base64)
      http-body-printable: yes # Dump HTTP body dạng readable text
      websocket-payload: yes # Dump payload WebSocket (nếu có)
      websocket-payload-printable: yes # Dump payload WebSocket (nếu có)

      metadata:
        app-layer: true       # Include decoded HTTP data
        flow: true            # Include flow state
      rule:
        metadata: true         # Log rule metadata
        raw: false             # Do not log raw rule text
        reference: false       # Do not log rule references
      ...
      ...
      ...
      ...

```

2.5. Thiếp lập Splunk cho việc theo dõi và phân tích logs

Về mặt kiến trúc, giải pháp Splunk được triển khai bằng cách cài đặt Splunk Universal Forwarder (UF) trên từng node trong cụm Kubernetes. Các agent (UF) này có nhiệm vụ thu thập và chuyển tiếp log (dựa trên cấu hình định sẵn) đến Splunk indexer, thành phần chịu trách nhiệm lưu trữ dữ liệu. Quá trình truy vấn và phân tích dữ liệu trên indexer sau đó được thực hiện thông qua Splunk search head.



- Trên tại các node thu thập các log hệ thống như auth.log, syslog,..v..v..

```

Monitored Files:
  $SPLUNK_HOME/etc/splunk.version
  /Library/Logs
  /root/.bash_history
  /var/adm
  /var/log/auth.log
  /var/log/suricata
  /var/log/syslog
ubuntu@ip-172-31-26-89:/opt/splunkforwarder/bin$ |

```

- Trên tại masternode có thêm log suricata giám sát các lưu lượng mạng vào ra.
- Trên tại indexer: log tập trung vào /var/log/fluent-bit-central/, tại đó indexer sẽ đọc vào file log được gửi tới và onboard.

```
Monitored Files:
$SPLUNK_HOME/etc/splunk.version
/Library/Logs
/root/.bash_history
/var/adm
/var/log/fluent-bit-central
ubuntu@ip-172-31-33-241:/opt/splunk/bin$ |
```

- Xem log thông qua giao diện web của Splunk

The screenshot shows the Splunk Enterprise search interface. The search bar contains the query `index=_source "/var/log/fluent-bit-central/logs.json"`. The results show 346,702 events matched. The interface includes various navigation and configuration buttons like Save As, Create Table View, Find, and Settings. The main area displays the search results in a table format with columns for Time, Event, and host. The host column shows entries like `host = ip-172-31-33-241 index = linux_log node_name = ip-172-31-16-170 source = /var/log/fluent-bit-central/logs.json`. The logs themselves contain detailed information such as log levels (INFO, WARNING), timestamps, and specific log messages related to fluent-bit operations like MTU detection and summary reconciliation loops.

Các log từ k8s gửi đến indexer và được lưu tại /var/log/fluent-bit-central/*.log, ngoài ra các log như syslog, auth.log hay suricata đè đã lên đầy đủ. Các thông tin đều được chuẩn hóa theo thời gian UTC, hỗ trợ cho việc điều tra số và lập CoC.

CHƯƠNG 3: THỰC NGHIỆM MÔ PHỎNG TẤN CÔNG VÀ ĐÁNH GIÁ

3.1. Mô phỏng tấn công vào ứng dụng web

Để kiểm chứng hiệu quả của pipeline đã xây dựng, nhóm đã thực thi các kịch bản tấn công nhắm vào ứng dụng web Juice Shop đang được triển khai trên cụm K8s. Burp Suite scanner, một scanner lỗ hổng web tổng hợp, được sử dụng cho giai đoạn initial access nhờ khả năng crawl và test với nhiều dạng lỗ hổng khác nhau. Báo cáo từ Burp Suite cho thấy bằng chứng về lỗ hổng SQL injection. Phân tích chỉ ra rằng tham số `?q=` tại đường dẫn `/rest/products/search` phản ứng không nhất quán khi nhận các probe chứa ký tự đặc biệt, qua đó xác nhận khả năng chèn cú pháp SQL.

The screenshot shows the Burp Suite interface with four panels:

- 4. Extension driven passive audit:** Shows 6 issues found.
- 3. Crawl and audit of juice.local:** Shows 5 issues found.
- 2. Live audit from Proxy (all traffic):** Shows 17 issues found.
- 1. Live passive crawl from Proxy (all traffic):** Shows 0 issues found.

The main content area displays a table of findings:

Time	Source	Issue type	Host	Path	Insertion point	Severity	Confidence
16:50:21 26 thg 10 20...	Task 3	Input returned in response (reflected)	https://juice.local	/api/Quantities/	URL path folder 1	Information	Certain
16:48:53 26 thg 10 20...	Task 3	SQLite injection	https://juice.local	/rest/products/search	q parameter	Medium	Firm
16:48:46 26 thg 10 20...	Task 3	Input returned in response (reflected)	https://juice.local	/rest/admin/application-configuration	URL path folder 1	Information	Certain
16:48:43 26 thg 10 20...	Task 3	Input returned in response (reflected)	https://juice.local	/rest/admin/application-version	URL path folder 1	Information	Certain
16:41:48 26 thg 10 20...	Task 3	Unencrypted communications	http://juice.local	/		Low	Certain
16:41:46 26 thg 10 20...	Task 3	OpenAPI definition found (active scan check)	https://juice.local	/api-docs/swagger-ui-imit.js		Information	Certain
16:41:45 26 thg 10 20...	Task 3	TLS certificate	https://juice.local	/		Medium	Certain
16:41:45 26 thg 10 20...	Task 3	Robots.txt file	https://juice.local	/robots.txt		Information	Certain

Details for the SQLite injection issue (index 164853):

- Severity: Medium
- Confidence: Firm
- URL: <https://juice.local/rest/products/search>

Note: This issue was generated by the Burp extension: Backslash Powered Scanner.

```
kali㉿kali: ~/Documents/difo-project [ ] kali㉿kali: ~/Documents/difo-project [ ]  
es so sqlmap could be able to run properly  
[06:09:08] [INFO] resuming back-end DBMS 'sqlite'  
[06:09:10] [INFO] testing connection to the target URL  
sqlmap resumed the following injection point(s) from stored session:  
Parameter: #1* (URI)  
Type: boolean-based blind  
Title: AND boolean-based blind - WHERE or HAVING clause  
Payload: https://juice.local/rest/products/search?q=' AND 2678=2678 AND ('hk0y' LIKE 'hk0y')  
Type: time-based blind  
Title: SQLite > 2.0 AND time-based blind (heavy query)  
Payload: https://juice.local/rest/products/search?q=' AND 3157=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2))) AND ('jsJv' LIKE 'jsJv')  
[06:09:11] [INFO] the back-end DBMS is SQLite  
back-end DBMS: SQLite  
[06:09:11] [INFO] fetching tables for database: 'SQLite_masterdb'  
[06:09:11] [INFO] fetching number of tables for database 'SQLite_masterdb'  
[06:09:11] [INFO] resumed: 21  
[06:09:11] [INFO] retrieving the length of query output  
[06:09:11] [INFO] resumed: 5  
[06:09:11] [INFO] resumed: Users  
[06:09:11] [INFO] retrieving the length of query output  
[06:09:11] [INFO] resumed: 15  
[06:09:11] [INFO] resumed: sqlite_sequence  
[06:09:11] [INFO] retrieving the length of query output  
[06:09:11] [INFO] resumed: 9  
[06:09:11] [INFO] resumed: Addresses  
[06:09:11] [INFO] retrieving the length of query output  
[06:09:11] [INFO] resumed: 7
```

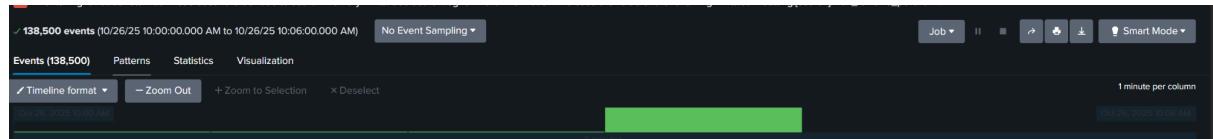
Từ điểm khởi đầu này, attacker tiến hành khai thác sâu hơn vào lỗ hổng được phát hiện bằng công cụ SQLmap. Công cụ sqlmap đã xác nhận endpoint `https://juice.local/rest/products/search?q=` có lỗ hổng và nhận diện hệ quản trị cơ sở

dữ liệu là SQLite. Điều này cho phép công cụ thu thập thông tin về cấu trúc cơ sở dữ liệu, cụ thể là bảng sqlite_master cùng các bảng dữ liệu như Users, Addresses,...

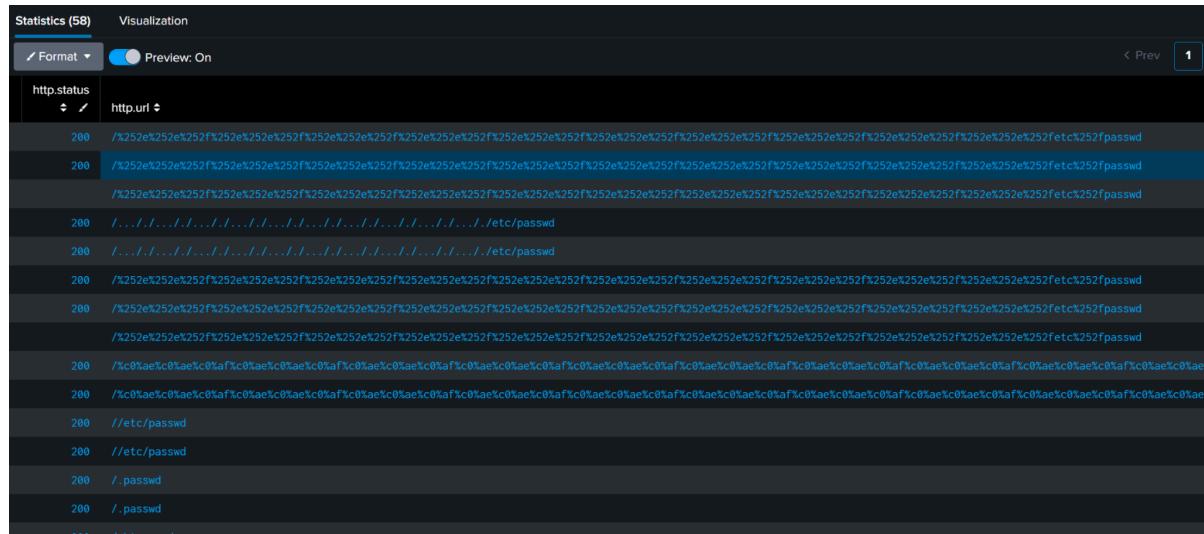
Nhóm mô phỏng hành vi khai thác nhằm đánh cắp dữ liệu trong database. Bằng việc dùng sqlmap, attacker thành công đánh cắp được nhiều thông tin trong ứng dụng và đồng thời cũng để lại nhiều logs liên quan đến traffic HTTP chứa dữ liệu độc hại. Các dữ liệu này được thu nhập và gửi đi liên tục trên pipeline bởi các pod chạy các agent của hai công cụ Fluent Bit và Suricata.

3.2. Đánh giá hiệu quả của pipeline và quy trình được đề xuất

Ngay tại thời điểm attacker khởi động tấn công bằng phương thức quét lỗ hổng có trên web app thì ngay lập tức, các logs từ node master được gửi liên tục tới Splunk indexer và trên dashboard thấy rằng có lượng lớn truy cập đến hệ thống. Điều này cho thấy pipeline mà nhóm đã xây dựng hoạt động trơn tru các logs lên kịp thời, đáp ứng được cho việc ứng phó sự cố kịp lúc làm giảm thời gian xử lý.



Trong khoảng thời gian từ 10/26/25 10:00:00.000 AM → 10/26/25 10:06:00.000 AM ghi nhận khoảng hơn 100 nghìn request trong khoảng 6 phút. Nhiều payload độc hại của kẻ tấn công cũng đã được khi nhận. Kẻ tấn công sử dụng các phương thức như: Path Traversal, SQL injection,... Từ đây có thể xác định được nguy hiểm tấn công của attacker, các payload hắn đã sử dụng và payload đã được thực thi thành công.



http.status	http.url
200	/rest/products/search? q=%27%29%20AND%20SUBSTR%28%28SELECT%20COALESCE%28tbl_name%2CCHAR%2832%29%29%20FROM%20sqlite_master%20WHERE%20type%3DCHAR%2811%2C97%2C98%2C108%
200	/rest/products/search? q=%27%29%20AND%20SUBSTR%28%28SELECT%20COALESCE%28tbl_name%2CCHAR%2832%29%29%20FROM%20sqlite_master%20WHERE%20type%3DCHAR%2811%2C97%2C98%2C108%
200	/rest/products/search?
200	/rest/products/search?q=
200	/rest/products/search? q=%27%29%20AND%20SUBSTR%28%28SELECT%20COALESCE%28tbl_name%2CCHAR%2832%29%29%20FROM%20sqlite_master%20WHERE%20type%3DCHAR%2811%2C97%2C98%2C108%
200	/rest/products/search?

Nếu không có pipeline thì phải chờ đến khi có sự cố ảnh hưởng tới hệ thống thì mới bắt đầu điều tra và việc điều tra cũng gặp nhiều khó khăn như thu thập logs từ các node, sao chép, phân tích thủ công, làm mất nhiều thời gian xử lý và khắc phục. Tóm lại, thông qua thực nghiệm, pipeline tự động thu thập chứng cứ được chứng minh hoạt động ổn định và đáng tin cậy. Các logic phát hiện và thu thập được thiết lập đã hoạt động chính xác, cho phép ứng phó sự cố an ninh theo thời gian thực.

PHẦN BA: KẾT LUẬN

1. Các kết quả đạt được

Đề tài đã hoàn thành mục tiêu cốt lõi là xây dựng một playbook và pipeline kỹ thuật tự động hóa, hỗ trợ đắc lực cho công tác điều tra pháp chứng trong môi trường Containerized Applications. Các kết quả chính đạt được bao gồm:

- ❖ **Xây dựng thành công pipeline thu thập tự động:** Đã thiết kế và triển khai một pipeline hoàn chỉnh, có khả năng tự động thu thập và tập trung hóa bằng chứng từ nhiều lớp trong môi trường Kubernetes. Pipeline này tích hợp các nguồn artifacts then chốt:
 - **Cấp độ Orchestration:** Kích hoạt và thu thập Kubernetes Audit Logs, cung cấp bằng chứng chi tiết về mọi hành vi tương tác với API server (ai, làm gì, khi nào).
 - **Cấp độ Host:** Triển khai và cấu hình auditd trên tất cả các node để giám sát các hoạt động nhạy cảm ở cấp độ hệ điều hành, như thay đổi tệp tin hệ thống hoặc truy cập socket của container runtime.
 - **Cấp độ Container & Network:** Sử dụng Fluent Bit (dưới dạng DaemonSet) để thu thập log của container, log hệ thống và Suricata để giám sát, ghi lại lưu lượng mạng đáng ngờ.
- ❖ **Thiết lập hệ thống phân tích và lưu trữ tập trung:** Toàn bộ dữ liệu từ các nguồn trên được chuyển tiếp an toàn và được thu thập, lập chỉ mục vào Splunk. Hệ thống Splunk đóng vai trò là kho lưu trữ bằng chứng trung tâm, cho phép chuẩn hóa thời gian và cung cấp giao diện mạnh mẽ để truy vấn, phân tích và tương quan dữ liệu.
- ❖ **Kiểm chứng qua thực nghiệm:** Pipeline đã được kiểm chứng tính hiệu quả thông qua kịch bản mô phỏng tấn công SQL injection vào ứng dụng web. Kết quả phân tích trên Splunk đã cho thấy pipeline thu thập thành công và rõ ràng các bằng chứng của cuộc tấn công, bao gồm các payload độc hại, chứng minh khả năng bảo tồn bằng chứng trong môi trường có tính tạm thời cao.

2. Những hạn chế

Hệ thống cũng bộc lộ một số nhược điểm đáng kể. Trở ngại lớn nhất nằm ở chi phí tài nguyên và lưu trữ. Do pipeline tạo ra một khối lượng dữ liệu khổng lồ từ auditd, K8s audit và Suricata nên sẽ đặt ra vấn đề về dung lượng lưu trữ trên Splunk indexer. Đồng thời, việc duy trì hoạt động của các agent (Splunk UF, Fluent Bit) cũng tiêu tốn tài nguyên tính toán (CPU/memory) đáng kể ngay trên các node của cluster.

Một thách thức khác là độ phức tạp trong triển khai và bảo trì. Việc cấu hình hoàn chỉnh pipeline đòi hỏi kiến thức chuyên môn sâu rộng về nhiều công nghệ riêng biệt, từ chính sách audit của Kubernetes, quy tắc của auditd trên Linux, các bộ lọc (parsers, filters) của Fluent Bit, cho đến việc quản lý dữ liệu đầu vào (data onboarding) và chỉ mục (index management) trên Splunk.

Cuối cùng, một hạn chế về phạm vi là pipeline hiện tại tập trung chủ yếu vào việc thu thập logs.. Hệ thống chưa tự động hóa quy trình thu thập các bằng chứng "sóng" khác, chẳng hạn như trích xuất bộ nhớ (memory dumps) hoặc tạo ảnh đĩa (filesystem snapshots) của container tại thời điểm xảy ra sự cố.

3. Hướng phát triển

Dựa trên các kết quả và hạn chế đã phân tích, hướng phát triển then chốt là tích hợp khả năng thu thập bằng chứng "sóng" một cách tự động. Hệ thống có thể được nâng cấp để bổ sung các cơ chế kích hoạt (trigger) nhằm phản ứng với các sự kiện an ninh nghiêm trọng. Chẳng hạn, khi Splunk phát hiện một cảnh báo ưu tiên cao (như reverse shell từ Suricata), nó có thể tự động khởi chạy một K8s Job để thực hiện snapshot bộ nhớ (memory dump) và sao chép hệ thống tệp (filesystem copy) của container bị nghi ngờ, đảm bảo thu giữ được bằng chứng dễ mất trước khi nó bị phá hủy.

Để giải quyết vấn đề về hiệu năng và chi phí tài nguyên của các agent, hướng nghiên cứu tiếp theo là ứng dụng eBPF cho giám sát runtime. Các công cụ như Falco hoặc Sysdig, vốn dựa trên eBPF, có thể được xem xét để thay thế hoặc bổ sung cho auditd. eBPF cung cấp khả năng quan sát cấp độ kernel sâu sắc với chi phí tài nguyên thấp hơn, đồng thời mang tính "cloud-native", khiến nó phù hợp tự nhiên hơn với môi trường Kubernetes.

Song song đó, việc tối ưu hóa khối lượng dữ liệu là một yêu cầu thực tiễn. Điều này bao gồm việc tinh chỉnh các chính sách audit để giảm thiểu "noise", chỉ tập

trung vào các sự kiện có giá trị pháp lý cao nhất, nhằm cân bằng giữa tính đầy đủ của bằng chứng và chi phí lưu trữ. Đồng thời, việc triển khai các chính sách quản lý vòng đời dữ liệu (data tiering) trong Splunk (như hot, warm, cold storage) là cần thiết để tối ưu hóa chi phí vận hành lâu dài.

Cuối cùng, hướng phát triển mang tính chiến lược nhất là nâng cấp pipeline từ vai trò "thu thập" (collection) lên "phản ứng" (response). Bằng cách tích hợp với các nền tảng SOAR (Security Orchestration, Automation, and Response), hệ thống có thể tự động thực thi các hành động khắc phục cơ bản, ví dụ như tự động áp dụng NetworkPolicy để cô lập pod bị xâm nhập, hoặc "đóng băng" (pause) container để bảo toàn trạng thái nguyên vẹn cho quá trình điều tra chi tiết.

TÀI LIỆU THAM KHẢO

- [1] Open Container Initiative. (October 2025). Online: <https://opencontainers.org/>
- [2] European Parliament & Council. (2016, April 27). Regulation (EU) 2016/679 of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). Official Journal of the European Union, L 119. <https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679>
- [3] National Commission on Forensic Science. (2015, December 7). Forensic science activities at NIST [Presentation]. U.S. Department of Justice. Retrieved October 26, 2025, from <https://www.justice.gov/ncfs/file/798951/dl?inline>
- [4] Malik, A. W., Bhatti, D. S., Park, T.-J., Ishtiaq, H. U., Ryou, J.-C., & Kim, K.-I. (2024). Cloud digital forensics: Beyond tools, techniques, and challenges. Sensors (Basel, Switzerland). Advance online publication. <https://doi.org/10.3390/s24020433>