THÔNG TIN CHUNG CỦA NHÓM

- Link YouTube video của báo cáo (tối đa 5 phút): https://youtu.be/PbSmAOMtSYo
- Link slides (dạng .pdf đặt trên Github của nhóm): *https://github.com/cuongth20-code/CS2205.CH201/blob/main/MO_HINH_BAO_MAT_RUNTIME_EBPF_VA_ML.pdf*

- Họ và Tên: Trương Hồng Cương
- MSSV: 250202004

- Lớp: CS2205.CH201
- Tự đánh giá (điểm tổng kết môn):8.5/10
- Số buổi vắng: 0
- Số câu hỏi QT cá nhân: 3
- Số câu hỏi QT của cả nhóm: 15
- LinkGithub: https://github.com/cuongth20-code/CS2205.CH201.git

ĐỀ CƯƠNG NGHIÊN CỨU

TÊN ĐỀ TÀI (IN HOA)

MÔ HÌNH BẢO MẬT RUNTIME DỰA TRÊN NGUYÊN TẮC ZERO TRUST SỬ DỤNG CÔNG NGHỆ EBPF VÀ HỌC MÁY (MACHINE LEARNING)

TÊN ĐỀ TÀI TIẾNG ANH (IN HOA)

A RUNTIME SECURITY MODEL BASED ON ZERO TRUST PRINCIPLE USING EBPF AND MACHINE LEARNING

TÓM TẮT

Đề tài "Mô hình bảo mật runtime dựa trên nguyên tắc Zero Trust sử dụng eBPF và học máy" tập trung xây dựng cơ chế giám sát và bảo vệ ở tầng runtime cho môi trường cloud-native (Docker/Kubernetes), nơi các hành vi tấn công có thể diễn ra ngay trong tiến trình/container và khó phát hiện bằng các giải pháp bảo mật dựa trên chữ ký hoặc rule tĩnh. Luận văn đề xuất mô hình Zero Trust runtime theo hướng đánh giá tin cậy liên tục (continuous verification) dựa trên quan sát hành vi thực thi.

Giải pháp kết hợp eBPF để thu thập dữ liệu sự kiện ở tầng kernel (syscall/process/network) và Machine Learning để phát hiện bất thường theo thời gian thực. Kiến trúc hệ thống gồm chuỗi thành phần: eBPF Collector → Data Processor → ML Engine → Trust Scoring → Policy/Enforcement. Dữ liệu thu thập được tiền xử lý, trích xuất đặc trưng hành vi theo tiến trình/container; sau đó mô hình học không giám sát (ví dụ Isolation Forest hoặc Autoencoder) học "hành vi bình thường" và phát hiện sai lệch. Kết quả bất thường được chuyển thành điểm tin cậy động (Dynamic Trust Score), làm cơ sở kích hoạt chính sách Zero Trust runtime như cảnh báo, hạn chế hành vi rủi ro hoặc cô lập container khi điểm tin cậy giảm dưới ngưỡng.

Mô hình được triển khai thử nghiệm trên Docker/Kubernetes và đánh giá theo các tiêu chí: độ chính xác phát hiện (Precision/Recall/F1), độ trễ xử lý, chi phí CPU/RAM, và so sánh với baseline từ công cụ giám sát runtime phổ biến (ví dụ Falco). Sản phẩm dự kiến gồm mã nguồn thử nghiệm, bộ log runtime eBPF và dashboard giám sát thời gian thực (Prometheus/Grafana), kèm khuyến nghị tích hợp vào quy trình DevSecOps/SOC.

GIỚI THIỆU

Trong những năm gần đây, kiến trúc cloud-native và microservices trở thành xu hướng chủ đạo trong xây dựng và vận hành hệ thống thông tin nhờ khả năng triển khai nhanh, mở rộng linh hoạt và tự động hóa theo mô hình DevSecOps. Việc sử dụng container và Kubernetes giúp tăng tính di động của ứng dụng và tối ưu tài nguyên, tuy nhiên cũng làm gia tăng đáng kể bề mặt tấn công ở tầng runtime—giai đoạn tiến trình/container đang thực thi. Nhiều hành vi tấn công hiện đại diễn ra ngay trong runtime như thực thi lệnh trái phép, tải và chạy mã độc, leo thang đặc quyền, truy cập secrets/volume, di chuyển ngang giữa dịch vụ và rò rỉ dữ liệu. Trong bối cảnh đó, các cơ chế bảo vệ truyền thống (IDS/IPS, WAF, chữ ký/rule tĩnh) thường gặp hạn chế do thiếu khả năng quan sát sâu hành vi thực thi, đồng thời khó thích ứng với sự biến động liên tục của workload trong môi trường container hóa.

Nguyên tắc Zero Trust nhấn mạnh "không tin mặc định" và yêu cầu xác minh liên tục dựa trên ngữ cảnh, danh tính và hành vi. Tuy nhiên, việc áp dụng Zero Trust ở tầng runtime vẫn còn khoảng trống do thiếu cơ chế thu thập tín hiệu hành vi đủ chi tiết và thiếu phương pháp định lượng mức tin cậy động cho tiến trình/container để hỗ trợ ra quyết định chính sách theo thời gian thực. Vì vậy, cần một mô hình bảo mật runtime có khả năng giám sát động, phát hiện bất thường không phụ thuộc chữ ký và hỗ trợ phản ứng kịp thời nhằm giảm thiểu rủi ro từ các biến thể tấn công mới hoặc lạm dụng quyền trong hệ thống.

Trong hướng tiếp cận này, eBPF (Extended Berkeley Packet Filter) là công nghệ phù hợp để quan sát sự kiện ở tầng kernel như syscall, process và network với độ chi tiết cao và chi phí tương đối thấp, tạo nền tảng cho việc theo dõi hành vi runtime của tiến trình/container. Kết hợp với Machine Learning, đặc biệt là các kỹ thuật phát hiện bất thường, hệ thống có thể học đặc trưng hành vi "bình thường" của workload và nhận diện sai lệch trong thời gian thực, từ đó hỗ trợ mô hình Zero Trust runtime theo hướng đánh giá liên tục. Trên cơ sở đó, đề tài "Mô hình bảo mật runtime dựa trên nguyên tắc Zero Trust sử dụng eBPF và học máy" hướng đến nghiên cứu–thiết kế–hiện thực một mô hình giám sát và bảo vệ runtime, trong đó kết quả phân tích hành vi được chuyển thành điểm tin cậy động (Dynamic Trust Score) cho từng tiến trình/container nhằm kích hoạt các chính sách phù hợp như cảnh báo, hạn chế hành vi rủi ro hoặc cô lập khi mức tin cậy giảm dưới ngưỡng. Mô hình được triển khai thử nghiệm trong Docker/Kubernetes và đánh giá theo độ chính xác phát hiện, độ trễ xử lý, chi phí tài nguyên và khả năng áp dụng trong thực tế.

MỤC TIÊU

Đề tài hướng tới việc nghiên cứu, thiết kế và hiện thực hóa một mô hình bảo mật runtime dựa trên nguyên tắc Zero Trust, kết hợp công nghệ eBPF và học máy nhằm tăng cường khả năng giám sát, phát hiện và phản ứng với các hành vi bất thường trong môi trường container hóa (Docker/Kubernetes). Cụ thể gồm 03 mục tiêu sau:

Khảo sát và xác định khoảng trống nghiên cứu

- Nghiên cứu cơ sở lý thuyết và các tiêu chuẩn/khung tham chiếu về Zero Trust (NIST SP 800-207, CISA ZTMM 2.0).
- Khảo sát các giải pháp giám sát và bảo mật runtime hiện có dựa trên rule/eBPF (ví dụ Falco, Tracee, Tetragon/Sysdig).
- Tổng hợp các phương pháp học máy phát hiện bất thường trong giám sát hệ thống (Isolation Forest, Autoencoder, …).
- Trên cơ sở đó, xác định khoảng trống kỹ thuật trong việc kết hợp telemetry eBPF với đánh giá tin cậy động và ra quyết định chính sách Zero Trust ở tầng runtime.

Đề xuất mô hình Zero Trust Runtime dựa trên eBPF và Machine Learning

- Thiết kế kiến trúc mô hình và ánh xạ rõ các thành phần theo Zero Trust (Policy Decision/Policy Enforcement), với pipeline:
  eBPF Collector → Data Processor → ML Engine → Trust Scoring → Policy Decision/Enforcement.
- Xây dựng phương pháp trích xuất đặc trưng hành vi tiến trình/container từ dữ liệu eBPF (syscall/process/network).
- Đề xuất cơ chế tính điểm tin cậy động (Dynamic Trust Score) dựa trên điểm bất thường (anomaly score) và ngữ cảnh runtime; xác định ngưỡng/luật quyết định để kích hoạt chính sách (cảnh báo, hạn chế, cô lập).

Hiện thực hóa và đánh giá thực nghiệm mô hình trong môi trường Docker/ Kubernetes

- Xây dựng prototype triển khai mô hình trên Docker/Kubernetes và tổ chức kịch bản kiểm thử gồm workload bình thường và các kịch bản tấn công điển hình.
- Đánh giá định lượng theo các tiêu chí: độ chính xác phát hiện (Precision/Recall/F1 hoặc ROC-AUC), độ trễ xử lý, overhead CPU/RAM, và khả năng mở rộng theo quy mô workload.
- So sánh kết quả với baseline từ công cụ runtime phổ biến (ví dụ Falco) để chứng minh hiệu quả và tính khả thi triển khai thực tế.

NỘI DUNG VÀ PHƯƠNG PHÁP

1. Nội dung nghiên cứu

Đề tài tập trung nghiên cứu và hiện thực hóa mô hình bảo mật runtime theo nguyên tắc Zero Trust, sử dụng eBPF để thu thập dữ liệu hành vi ở tầng kernel và áp dụng học máy phát hiện bất thường nhằm tính toán điểm tin cậy động (Dynamic Trust Score) cho tiến trình container. Các nội dung chính gồm:

1.1. Khảo sát cơ sở lý thuyết và hiện trạng giải pháp

- Nghiên cứu Zero Trust Architecture và cơ chế đánh giá liên tục ở tầng runtime.
- Khảo sát cơ chế hoạt động eBPF và các giải pháp runtime security hiện có (rule-based/eBPF).
- Tổng hợp các phương pháp học máy phát hiện bất thường phù hợp với dữ liệu telemetry hệ thống.

1.2. Thiết kế mô hình và kiến trúc hệ thống

- Đề xuất pipeline: eBPF Collector → Data Processor → ML Engine → Trust Scoring → Policy Decision/Enforcement.
- Xác định luồng dữ liệu, ngữ cảnh (container/pod/service), và cơ chế ra quyết định chính sách theo điểm tin cậy.
- Thiết kế cơ chế phản ứng theo mức độ rủi ro: cảnh báo/hạn chế/cô lập.

1.3. Xây dựng tập đặc trưng hành vi "core" (Feature set) từ eBPF

Để tránh lan man và đảm bảo tính khả thi, đề tài tập trung vào nhóm đặc trưng cốt lõi, đủ phân biệt bất thường nhưng không gây quá tải tính toán. Feature được chuẩn hóa theo đối tượng quan sát (process/container) và theo cửa sổ thời gian (time window). Các nhóm core gồm:

- **Syscall behavior (cốt lõi)**
  - Tần suất syscall theo nhóm (exec/file/net/process/memory).
  - Tỷ lệ syscall "nhạy cảm" (execve, ptrace, mount, setns, chmod/chown, …).
  - Độ đa dạng syscall (syscall diversity) và phân bố tần suất (ví dụ entropy).
- **Process behavior (cốt lõi)**
  - Số lần tạo tiến trình con (fork/clone rate), cây tiến trình bất thường.
  - Hành vi thực thi: số lần exec, đường dẫn binary, thay đổi user/group, thay đổi capabilities.
  - Chỉ báo "LOLBins" đơn giản (bash/sh/python/curl/wget xuất hiện đột biến trong container vốn không cần).
- **File behavior (cốt lõi)**
  - Số lần truy cập/ghi vào đường dẫn nhạy cảm (secrets, /etc, kube config, volume quan trọng).
  - Tỷ lệ read/write/rename/unlink; phát hiện "binary drop" (tạo file thực thi mới).
- **Network behavior (cốt lõi)**
  - Số kết nối ra ngoài, destination port/service lạ, outbound spike theo window.
  - DNS query rate bất thường; số endpoint mới xuất hiện.

Ghi chú: Các feature nâng cao (packet size distribution, syscall sequence n-gram sâu, embedding) được xem là mở rộng; chỉ triển khai khi đáp ứng giới hạn latency/overhead.

1.4. Xây dựng và huấn luyện mô hình phát hiện bất thường

- Chuẩn hóa dữ liệu theo window (ví dụ 5s/10s/30s) để tạo vector đặc trưng.
- Huấn luyện mô hình không giám sát (Isolation Forest, Autoencoder) trên dữ liệu "bình thường" theo từng workload hoặc từng service.
- Đầu ra mô hình là anomaly score, dùng làm đầu vào tính Dynamic Trust Score.

1.5. Xử lý Concept Drift trong môi trường runtime

Do workload thay đổi theo thời gian (scale up/down, deploy phiên bản mới, thay đổi traffic), đề tài bổ sung cơ chế xử lý concept drift nhằm giảm false positive và duy trì độ ổn định mô hình. Ba chiến lược được áp dụng:

- Sliding Window Update: cập nhật thống kê baseline và phân phối feature theo cửa sổ trượt; phát hiện sai lệch so với "bình thường gần đây".
- Periodic Retraining: tái huấn luyện theo chu kỳ (ví dụ theo ngày/tuần hoặc sau mỗi lần deploy), có cơ chế chọn dữ liệu "sạch" để tránh học nhầm hành vi tấn công.
- Per-service/Per-image Model: huấn luyện mô hình riêng cho từng service/image (hoặc namespace) thay vì một mô hình chung toàn cụm, nhằm phản ánh đúng đặc thù hành vi và giảm nhiễu.

1.6. Tính điểm tin cậy động và thực thi chính sách

- Chuyển anomaly score thành Dynamic Trust Score theo thời gian, có cơ chế làm mượt (smoothing) và ngưỡng/hysteresis để tránh dao động.
- Kích hoạt policy theo mức: Alert → Restrict → Isolate (ví dụ hạn chế outbound, chặn hành vi exec bất thường, cô lập pod/container).

1.7. Thực nghiệm và đánh giá

- Triển khai prototype trên Docker/Kubernetes; xây dựng workload bình thường và kịch bản tấn công điển hình.
- Đánh giá theo: Precision/Recall/F1 (hoặc ROC-AUC), latency end-to-end, CPU/RAM overhead, khả năng mở rộng theo số pod/node.
- So sánh baseline với công cụ runtime phổ biến (ví dụ Falco rule-based) để xác định mức cải thiện.

2. Phương pháp thực hiện

- Phân tích – tổng hợp tài liệu: Zero Trust, eBPF runtime security, anomaly detection.
- Mô hình hóa và thiết kế hệ thống: thiết kế kiến trúc pipeline, luồng dữ liệu, chính sách.
- Thực nghiệm định lượng: triển khai thử nghiệm, thu thập số liệu, phân tích thống kê.
- Huấn luyện và đánh giá ML: scikit-learn/PyTorch; tối ưu tham số; đánh giá theo chỉ số và theo kịch bản.

KẾT QUẢ MONG ĐỢI

Kết quả học thuật

- Mô hình bảo mật runtime theo nguyên tắc Zero Trust**:** Xây dựng và chứng minh một kiến trúc bảo mật mới dựa trên việc kết hợp công nghệ eBPF và học máy không giám sát (Unsupervised ML) để phát hiện hành vi bất thường và tính điểm tin cậy động (Dynamic Trust Score) cho từng tiến trình/container.
- Phân tích và đánh giá hiệu năng mô hình: Đánh giá định lượng mô hình qua các tiêu chí:
    - Độ chính xác phát hiện bất thường (Precision, Recall, F1-score).
    - Độ trễ xử lý (Latency).
    - Chi phí tài nguyên (CPU, Memory Overhead).
    - Khả năng mở rộng trong môi trường container.
- Đóng góp học thuật: Đề xuất phương pháp định lượng điểm tin cậy runtime và quy trình tích hợp mô hình vào kiến trúc Zero Trust, bổ sung vào hướng nghiên cứu "Adaptive Trust and Behavior-based Zero Trust Runtime Security".

Kết quả thực tiễn / sản phẩm đầu ra

- Mã nguồn thử nghiệm: Bộ mã nguồn mở (Python/BCC/eBPF) triển khai pipeline gồm:
    - eBPF Collector thu thập dữ liệu hành vi.
    - Data Processor tiền xử lý đặc trưng.
    - ML Engine huấn luyện và dự đoán.
    - Policy Layer thực thi chính sách runtime.
- Bộ dữ liệu runtime eBPF: Gồm tập log hành vi tiến trình và container được thu thập từ hệ thống thử nghiệm, có thể dùng làm benchmark cho các nghiên cứu sau.
- Dashboard giám sát thời gian thực: Xây dựng giao diện Prometheus /Grafana hiển thị điểm tin cậy runtime, biểu đồ phát hiện bất thường, và cảnh báo tự động theo thời gian thực