Abordagem Cross-layer para Detecção de Intrusão Integrando eBPF e Machine Learning

Daniel Arioza, Jeferson Campos Nobre, Lisandro Zambenedetti Granville

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

`{daniel.almeida,jcnobre,granville}@inf.ufrgs.br`

Abstract. *The increasing sophistication of multi-vector attacks challenges computational security, making efficient correlation between network and system events on a single host a critical problem. This work introduces CrossLayerGuardian, a hybrid intrusion detection system (IDS) with an eBPF-based architecture for efficient collection and cross-layer correlation. Unlike hybrid approaches that merely combine data sources, our cross-layer approach focuses on active and temporally-aware correlation between network and system events to identify complex attacks within a single host. The solution integrates high-performance network processing (XDP) and granular syscall monitoring, synchronized by kernel timestamps to ensure causal ordering. Correlated events are analyzed by an adaptive machine learning ensemble, featuring in-kernel pre-filtering to reduce overhead. Experimental evaluation demonstrated a throughput of 850 Mbps with CPU overhead below 8%, comparable to state-of-the-art eBPF-based systems. CrossLayerGuardianachieved detection rates exceeding 95% for both network and multi-vector attacks, with false positive rates below 1.2%. The results confirm that the proposed architecture provides an efficient solution for single-host cross-domain correlation, balancing performance and accuracy.*

Resumo. *A crescente sofisticação de ataques multi-vetor desafia a segurança computacional, tornando a correlação eficiente entre eventos de rede e sistema em um mesmo host um problema crítico. Este trabalho apresenta o CrossLayerGuardian, um sistema híbrido de detecção de intrusão (IDS) com arquitetura baseada em eBPF para coleta eficiente e correlação cross-layer. Diferente de abordagens híbridas que apenas combinam fontes de dados, nossa abordagem cross-layer foca na correlação ativa e temporalmente consciente entre eventos de rede e sistema para identificar ataques complexos dentro de um único host. A solução integra processamento de rede de alto desempenho (XDP) e monitoramento granular de syscalls, sincronizados por timestamps de kernel para garantir ordenação causal. Eventos correlacionados são analisados por um ensemble adaptativo de aprendizado de máquina, com pré-filtragem no kernel para reduzir overhead. A avaliação experimental demonstrou throughput de 850 Mbps com overhead de CPU inferior a 8%, comparável a sistemas estado da arte baseados em eBPF. O CrossLayerGuardianalcançou taxas de detecção superiores a 95% para ataques de rede e multi-vetor, com falsos positivos abaixo de 1,2%. Os resultados confirmam que a arquitetura proposta oferece uma solução eficiente para correlação cross-domain em host único, equilibrando desempenho e precisão.*

1. Introdução

A Cibersegurança enfrenta desafios sem precedentes com ataques computacionais evoluindo em frequência e sofisticação. Análises recentes indicam que 83% dos ataques bem-sucedidos em 2023 exploram múltiplos vetores coordenadamente [for Internet Security 2023], e 76% combinam técnicas de rede e sistema [Business 2023]. Esse cenário é agravado pelo tempo médio de detecção de 277 dias [Security 2023] e pelo aumento de 43% na complexidade dos ataques [for Cybersecurity 2023], gerando prejuízos médios de US$ 4.35 milhões por incidente.

Embora Sistemas de Detecção de Intrusão (IDS) sejam cruciais, as tecnologias atuais lutam contra ataques multi-vetor. Sistemas baseados em rede (NIDS), como Snort e Suricata, analisam bem o tráfego, mas carecem de visibilidade do host. Inversamente, sistemas baseados em host (HIDS), como Falco, monitoram o sistema localmente, mas perdem o contexto da rede. Uma meta-análise [Khraisat et al. 2019] confirma que essa fragmentação cria lacunas na detecção de ataques modernos. Tentativas de soluções híbridas frequentemente falham na sincronização e correlação eficiente entre camadas, gerando sobrecarga.

A tecnologia Extended Berkeley Packet Filter (eBPF) surge como promissora para superar tais limitações, permitindo monitoramento eficiente [Findlay 2020], redes neurais no kernel [Zhang et al. 2023] e integração com aprendizado de máquina [Kostopoulos 2023]. Contudo, soluções eBPF existentes ainda tratam rede e sistema isoladamente, sem resolver o problema fundamental da correlação cross-layer eficiente entre eventos dessas camadas em um mesmo host.

Neste contexto, apresentamos o CrossLayerGuardian, um IDS híbrido com capacidade cross-layer, projetado para a proteção de host único. Enquanto sistemas *híbridos* combinam fontes de dados, nossa abordagem *cross-layer* implementa correlação ativa e temporalmente consciente entre eventos de rede e sistema operacional para identificar ataques multi-vetor complexos no host. O CrossLayerGuardian propõe uma arquitetura coesa, expandindo conceitos anteriores [Findlay 2020, Kostopoulos 2023] com estruturas de dados compartilhadas entre kernel e userspace. Utiliza um ensemble adaptativo XGBoost+MLP no userspace para analisar eventos correlacionados, superando limitações de ML no kernel [Zhang et al. 2023]. Incorpora pré-filtragem eficiente no kernel via mapas eBPF e classificação em dois estágios [Chen et al. 2023], otimizados por XDP, além de agregação de syscalls [Byrnes et al. 2023] e amostragem adaptativa [Song et al. 2023] para monitoramento sistêmico.

A eficácia da solução é suportada por inovações como ring buffers otimizados [Wang and Chang 2023] e sincronização temporal de baixo impacto [Quincozes et al. 2021], e validada por resultados empíricos. A avaliação experimental em ambiente simulado (AWS EC2) demonstrou desempenho comparável a sistemas eBPF estado da arte: throughput de 850 Mbps com overhead de CPU inferior a 8%. O sistema alcançou taxas de detecção superiores a 95% para ataques de rede e multi-vetor, com falsos positivos abaixo de 1.2%, representando uma melhoria substancial sobre as limitações atuais.

As principais contribuições deste trabalho são: (1) uma arquitetura unificada baseada em correlação cross-layer para host único, abordando lacunas de [Khraisat et al. 2019, Findlay 2020]; (2) algoritmos otimizados de pré-filtragem no kernel com perfect hashing e estruturas lock-free para menor latência e maior escalabilidade; (3) suporte à sincronização temporal precisa entre eventos de rede e sistema, resolvendo desafios de [Zhang et al. 2023]; e (4) validação experimental robusta, conforme sugerido por [Vaishali 2023, Madhavi and Nethravathi 2023].

O restante deste artigo está organizado da seguinte forma: Seção 2 apresenta a fundamentação teórica e trabalhos relacionados. Seção 3 detalha a arquitetura e implementação do CrossLayerGuardian. Seção 4 descreve a metodologia de avaliação e analisa os resultados. Seção 5 conclui o trabalho e aponta direções futuras.

2. Trabalhos Relacionados e Conceitos Fundamentais

2.1. Evolução da Literatura em Sistemas de Detecção de Intrusão

A literatura sobre Sistemas de Detecção e Prevenção de Intrusão (IDPS) evoluiu significativamente na última década, transitando de abordagens baseadas em assinaturas para técnicas baseadas em comportamento e aprendizado de máquina. As pesquisas recentes focam principalmente em três vertentes: otimização de desempenho através de instrumentação de kernel [Findlay 2020, Hadi et al. 2023], aplicação de técnicas avançadas de ML para melhorar a precisão [Zhang et al. 2022, Madhavi and Nethravathi 2023], e desenvolvimento de arquiteturas híbridas que correlacionam eventos de rede e sistema [Wang and Lu 2020, Chen et al. 2023].

A integração eficiente entre dados de rede e host permanece um desafio central, com diversos trabalhos propondo soluções parciais. Enquanto [Wang and Lu 2020] demonstrou que a correlação cross-layer pode aumentar significativamente a detecção de ataques complexos, seus modelos exigem recursos computacionais proibitivos para ambientes de produção. Paralelamente, [Findlay 2020] e [Zhang et al. 2023] exploraram o potencial do eBPF para monitoramento de baixo overhead, mas limitaram-se a análises de camada única.

Esta revisão examina os principais avanços nestas áreas, identificando lacunas na literatura atual que o CrossLayerGuardian busca preencher, especialmente no que tange à correlação eficiente de eventos entre múltiplas camadas em tempo real com overhead mínimo.

2.2. Tecnologias Fundamentais

2.2.1. Extended Berkeley Packet Filter (eBPF)

O Extended Berkeley Packet Filter (eBPF) permite a execução segura de programas no kernel Linux, fornecendo um ambiente verificável com garantias de segurança [eBPF Community 2021]. Sua arquitetura utiliza verificação estática para garantir terminação, prevenir acessos inválidos à memória e impor limites de uso de recursos [Findlay 2020].

O eBPF é crucial para sistemas de segurança por oferecer acesso direto a estruturas do kernel, mapas para compartilhamento eficiente de estado e hooks para monitoramento de eventos. O XDP (eXpress Data Path) permite processamento de pacotes de rede em estágios iniciais do driver, antes da pilha de rede tradicional, possibilitando desempenho significativamente superior [Wang and Lu 2020].

Apesar das vantagens, o eBPF apresenta limitações para implementações complexas de ML no kernel: memória restrita (512 bytes por pilha), limite de instruções (1 milhão), suporte limitado a operações de ponto flutuante, restrições de loop, impossibilidade de utilizar bibliotecas otimizadas (BLAS, TensorFlow) e dificuldade na depuração. Estas limitações motivam abordagens híbridas que mantêm coleta eficiente no kernel, realizando análises complexas no userspace.

2.2.2. Aprendizado de Máquina para Detecção de Intrusão

Sistemas de detecção baseados em ML têm demonstrado efetividade, especialmente com abordagens ensemble [Zhang et al. 2022]. Combinações de XGBoost e MLPs (Multilayer Perceptrons) alcançam taxas de detecção superiores a 95% com baixos falsos positivos [Vaishali 2023, Madhavi and Nethravathi 2023].

O XGBoost é eficiente no processamento de datasets desbalanceados e features não-lineares [XGBoost 2022], enquanto MLPs são eficazes na detecção de padrões temporais em dados de segurança [Scikit-learn 2014]. A classificação em múltiplos estágios implementa detecção em duas fases: análise rápida inicial seguida por processamento profundo para casos ambíguos [Chen et al. 2023].

Para operação em tempo real, são essenciais técnicas como cache de features, inferência em lote e quantização de modelos [Kostopoulos 2023]. A efetividade depende da qualidade das features, incluindo características comportamentais de syscalls, padrões de rede e métricas de sistema.

2.3. Trabalhos Relacionados

2.3.1. Sistemas Baseados em eBPF

O trabalho de [Findlay 2020] estabeleceu as bases para utilização de eBPF em sistemas de detecção de intrusão, demonstrando instrumentação eficiente do kernel para monitoramento de segurança. Apesar de alcançar overhead significativamente menor que sistemas tradicionais, a implementação limitava-se a análises baseadas em regras simples, inadequada para detectar padrões de ataque complexos e adaptativos.

[Zhang et al. 2023] apresentou uma inovação ao implementar redes neurais diretamente no kernel através de eBPF, demonstrando capacidades de detecção em tempo real com latência média de 100 µs. Contudo, as restrições do eBPF limitaram a complexidade dos modelos, comprometendo a detecção de ataques sofisticados e desconhecidos.

O sistema iKern de [Hadi et al. 2023] alcançou throughput de até 10Gbps com baixo overhead de CPU utilizando eBPF em conjunto com PF_RING para captura otimizada de pacotes. Porém, seu foco exclusivo em análise de rede o torna vulnerável a ataques multi-vetor que combinam exploração de rede e sistema.

[Kostopoulos 2023] desenvolveu um framework combinando eBPF com ML para detecção em tempo real. Embora tenha demonstrado a viabilidade de modelos ML mais sofisticados operando em conjunto com eBPF, não abordou a correlação efetiva entre eventos de rede e sistema, essencial para detectar ataques complexos multi-vetor.

2.3.2. Abordagens Híbridas e Cross-Layer

[Wang and Lu 2020] propôs uma arquitetura integrando análise de host e rede usando deep learning, alcançando taxas de detecção superiores a 95% para ataques conhecidos. Apesar do potencial demonstrado, o alto overhead computacional e a complexidade de implantação tornam esta abordagem impraticável para ambientes de produção de larga escala.

A contribuição de [Chen et al. 2023] introduziu um sistema de classificação em dois estágios, combinando análise rápida de features com processamento profundo seletivo. Embora esta abordagem tenha equilibrado efetivamente performance e precisão, ela carece de mecanismos eficientes para correlação temporal de eventos entre diferentes camadas, limitando sua eficácia para ataques multi-vetor.

2.3.3. Análise de Syscalls e Correlação

No domínio de análise de syscalls, [Byrnes et al. 2023] apresentou uma implementação focada em sequências de chamadas de sistema, inovando na aplicação de técnicas de processamento de sequência para detecção de comportamentos anômalos. Embora tenha alcançado alta precisão em detecção de ataques baseados em host, o throughput limitado de 336 Mbps torna esta solução inadequada para redes modernas de alta velocidade.

[Song et al. 2023] explorou a análise baseada em sequências de syscalls para segurança de contêineres, demonstrando como padrões de chamadas de sistema podem indicar comportamentos maliciosos. Embora tenha fornecido insights valiosos sobre técnicas de correlação temporal, o foco exclusivo em ambientes containerizados limita a aplicabilidade da solução em infraestruturas diversificadas.

2.4. Limitações das Abordagens Existentes

A análise da literatura revela deficiências significativas nos sistemas de detecção atuais. A fragmentação entre soluções de rede (NIDS) e host (HIDS) cria vulnerabilidades exploráveis, especialmente contra ataques multi-vetor que atravessam fronteiras entre camadas de aplicação.

Sistemas existentes enfrentam um dilema fundamental entre precisão e desempenho. Implementações no kernel sofrem restrições computacionais severas, enquanto abordagens em userspace apresentam problemas de latência e sincronização. As avaliações experimentais demonstram que soluções híbridas frequentemente não conseguem manter análise efetiva em ambientes de alto throughput sem comprometer capacidade analítica.

A falta de adaptabilidade constitui outra limitação crítica. O monitoramento uniforme independente do contexto ou nível de risco resulta em alocação ineficiente de recursos, comprometendo tanto eficiência quanto segurança. Adicionalmente, mecanismos de correlação temporal entre eventos de diferentes camadas permanecem rudimentares, impossibilitando o estabelecimento de relações causais precisas.

O CrossLayerGuardian supera estas limitações através de uma arquitetura integrada que combina pré-filtragem eficiente no kernel via eBPF, comunicação otimizada entre camadas, análise adaptativa em userspace e correlação cross-layer com sincronização temporal precisa.

Tabela 1. Comparação Detalhada dos Trabalhos Relacionados

Trabalho	Escopo	Técnica Principal	K/U Space	Overhead	Throughput	Modelo ML	Limitações	Validação
[Findlay 2020]	HIDS	eBPF/syscalls	Kernel	< 5% CPU	N/A	Regras	Sem análise de rede	Benchmarks microk.
[Zhang et al. 2023]	NIDS	NN no kernel	Kernel	< 8% CPU	1Gbps	Single-layer NN	Modelo ML limitado	Testbed + CAIDA
[Hadi et al. 2023]	NIDS	eBPF+PF_RING	Híbrido	< 10% CPU	10Gbps	N/A	Apenas análise de rede	Enterprise real
[Kostopoulos 2023]	NIDS	eBPF+ML	Híbrido	< 15% CPU	1Gbps	Random Forest	Sem correlação host	Dataset sintético
[Wang and Lu 2020]	Híbrido	Deep Learning	User	> 20% CPU	100Mbps	CNN+LSTM	Alto overhead	CICIDS2017
[Chen et al. 2023]	HIDS	Two-stage	User	< 12% CPU	N/A	Ensemble	Sem contexto de rede	Ambiente virtual
[Byrnes et al. 2023]	HIDS	Syscall seq.	Híbrido	< 7% CPU	336Mbps	HMM	Throughput limitado	Traces reais
[Song et al. 2023]	HIDS	Seq. Analysis	Kernel	< 5% CPU	N/A	N/A	Apenas contêineres	Container testbed
CrossLayerGuardian	Cross-layer	eBPF+Ensemble	Híbrido	< 8% CPU	850 Mbps	XGBoost+MLP	Requer eBPF	Testbed+Prod+CICIDS

3. Arquitetura e Implementação

O CrossLayerGuardian implementa uma solução integrada para monitoramento e detecção de intrusões através de múltiplas camadas. Esta seção apresenta sua arquitetura, componentes fundamentais e aspectos técnicos de implementação.

3.1. Visão Geral

A arquitetura do CrossLayerGuardian, ilustrada na Figura 1, divide-se em componentes de kernel e userspace que operam de forma coordenada. O fluxo de dados inicia-se com a captura paralela de eventos de rede pelo NetMonitor e eventos de sistema pelo FileMonitor. Estes eventos são pré-processados no kernel para redução de volume e então transmitidos através de ring buffers para componentes de userspace. O EventCorrelator recebe estes fluxos de dados, realiza sincronização temporal e estabelece relações causais entre eventos de diferentes domínios. Finalmente, o Detector analisa os eventos correlacionados para identificação de comportamentos maliciosos.

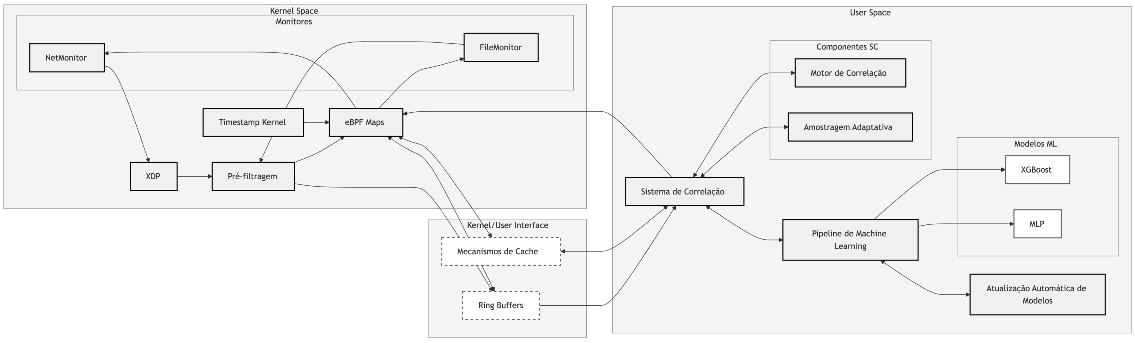
Kernel Space
Monitores
NetMonitor
FileMonitor
Timestamp Kernel
eBPF Maps
XDP
Pré-filtragem
Kernel/User Interface
Mecanismos de Cache
Ring Buffers
User Space
Componentes SC
Motor de Correlação
Amostragem Adaptativa
Sistema de Correlação
Modelos ML
XGBoost
MLP
Pipeline de Machine Learning
Atualização Automática de Modelos

Figura 1. Arquitetura do CrossLayerGuardian mostrando componentes kernel e userspace

3.2. Componentes Principais

3.2.1. NetMonitor

O NetMonitor captura e analisa tráfego de rede através de dois níveis complementares de processamento. O filtro XDP opera diretamente no driver de rede antes do processamento pela pilha de rede do kernel. Esta implementação realiza três operações principais descarregadas via XDP: verificação de assinaturas básicas contra listas de bloqueio, validação estrutural de protocolos para eliminar pacotes malformados e controle de taxa para mitigar ataques de inundação.

As regras de pré-filtragem no kernel incluem verificações estáticas de portas e endereços conhecidos como maliciosos, padrões de bytes específicos em posições fixas de cabeçalhos e verificações de validade de protocolos. Estas regras são deliberadamente limitadas em complexidade para garantir processamento de alta velocidade, com foco em reduzir o volume de dados que requer análise mais profunda.

O mecanismo de rastreamento de fluxo mantém estatísticas de conexões utilizando mapas hash eBPF. Para cada fluxo, o sistema rastreia contadores de pacotes e bytes segregados por direção, distribuição de tamanhos de pacote, intervalos entre pacotes e flags de controle TCP observadas. Esta implementação utiliza uma estrutura de tupla de cinco elementos como chave (IP origem, IP destino, porta origem, porta destino, protocolo) e uma estrutura de dados customizada como valor para armazenamento eficiente das estatísticas coletadas.

3.2.2. FileMonitor

O FileMonitor monitora operações de sistema de arquivos e chamadas de sistema através de kprobes e tracepoints. O sistema captura eventos como modificações em arquivos críticos, execução de processos, operações de privilégios elevados e acesso a recursos sensíveis.

O mecanismo de agregação de syscalls implementa três níveis de processamento. Primeiro, syscalls relacionadas são agrupadas em operações lógicas (por exemplo, uma sequência de open(), read(), write(), close() é consolidada em uma operação de ”modificação de arquivo”). Segundo, operações repetitivas sobre o mesmo recurso são sumarizadas com contadores e timestamps de início/fim. Terceiro, operações relacionadas ao mesmo contexto semântico (por exemplo, múltiplas leituras sequenciais de arquivos de configuração) são agrupadas em eventos de alto nível denominados ”sessões de atividade”.

O monitoramento adaptativo funciona através de um mecanismo de pontuação dinâmica. Cada processo recebe uma pontuação de risco inicial baseada em características estáticas como caminho de execução, usuário executor e argumentos de linha de comando. Esta pontuação é continuamente ajustada com base no comportamento observado, como acesso a recursos sensíveis, comunicação com endereços externos ou execução de operações privilegiadas. A intensidade de monitoramento é modulada proporcionalmente à pontuação de risco, com três níveis definidos: baixo (amostragem de 10% das syscalls), médio (amostragem de 50% com rastreamento de eventos específicos) e alto (monitoramento completo com captura detalhada de parâmetros e valores de retorno).

3.2.3. Limitações Técnicas do eBPF para ML no Kernel

A implementação de algoritmos complexos de ML diretamente no kernel através de eBPF enfrenta limitações técnicas significativas. Primeiramente, programas eBPF são limitados a

4096 instruções BPF após compilação (expandido para 1 milhão em kernels recentes, mas ainda insuficiente para modelos complexos). Esta restrição impossibilita a implementação direta de redes neurais ou árvores de decisão profundas no kernel.

Segundo, eBPF não suporta nativamente operações de ponto flutuante necessárias para a maioria dos algoritmos de ML. Embora simulações de ponto flutuante sejam possíveis, elas incorrem em overhead significativo e complexidade adicional. Adicionalmente, programas eBPF possuem um limite estrito de 512 bytes de espaço de pilha, insuficiente para estruturas de dados intermediárias exigidas por algoritmos de ML.

O modelo de memória eBPF também impõe desafios, pois o acesso a dados é primariamente realizado através de mapas eBPF, que têm limitações de tamanho e flexibilidade comparados às estruturas de dados disponíveis em userspace. Por fim, a validação estática do verificador eBPF rejeita construções complexas como loops não desenrolados, fundamentais para muitos algoritmos de ML. Estas limitações justificam a decisão arquitetural de realizar pré-filtragem simples no kernel enquanto reservar algoritmos de ML complexos para a execução em userspace.

3.2.4. Ring Buffers e Sincronização Inter-CPU

O sistema utiliza ring buffers eBPF para comunicação eficiente entre componentes kernel e userspace. A implementação emprega o mecanismo BPF_RINGBUF introduzido em kernels Linux 5.8+, que oferece sincronização lock-free entre produtores e consumidores multi-thread.

Cada ring buffer é dimensionado dinamicamente no momento da inicialização do sistema, com tamanho baseado na memória disponível e no volume esperado de eventos, tipicamente entre 4MB e 16MB por buffer. Em caso de overflow iminente (quando consumo é mais lento que produção), o sistema implementa um mecanismo de descarte seletivo baseado em prioridade de eventos, onde eventos considerados menos críticos para análise de segurança são descartados primeiro, preservando eventos com maior potencial de detecção.

Os ring buffers armazenam objetos de evento serializados contendo metadados comuns (timestamp, tipo de evento, identificadores de processo), dados específicos ao tipo de evento e chaves de correlação para relacionamento com outros eventos. Para eventos de rede, o conteúdo inclui tuplas de conexão, estatísticas de fluxo e flags observadas. Para eventos de sistema, são armazenados identificadores de recursos, operações realizadas e resultados relevantes.

A sincronização inter-CPU representa um desafio significativo, pois eventos de rede e sistema podem ser capturados em diferentes CPUs com potenciais variações de timestamp. O sistema implementa um mecanismo de sincronização baseado em TSC (Time Stamp Counter) normalizado, que compensa diferenças entre contadores de ciclos em CPUs distintas. Adicionalmente, cada evento recebe um identificador monótono de sequência dentro de seu domínio, permitindo detecção de desvios de ordem causal mesmo quando timestamps apresentam pequenas discrepâncias.

3.2.5. EventCorrelator

O EventCorrelator estabelece relações temporais e causais entre eventos de rede e sistema. No contexto do CrossLayerGuardian, um ”evento”é formalmente definido como uma tupla $E = (\tau, \delta, \alpha, \rho, \kappa)$, onde τ representa o timestamp preciso de ocorrência, δ identifica o domínio de origem (rede ou sistema), α descreve a ação observada, ρ contém os recursos envolvidos e κ representa o conjunto de chaves de correlação que permitem associação com outros eventos relacionados.

A correlação temporal entre eventos é implementada utilizando um algoritmo de janela deslizante adaptativa. Este algoritmo mantém janelas de correlação ajustadas dinamicamente com base nas características observadas no sistema monitorado. A necessidade desta correlação deriva do fato de que ataques multi-vetor manifestam-se através de padrões temporalmente relacionados que não seriam detectáveis analisando eventos isoladamente ou em domínios separados.

O mecanismo de sincronização por timestamp de kernel utiliza a função clock_gettime(CLOCK_MONOTONIC) como fonte primária de temporização, aplicando a marcação temporal no ponto mais próximo possível à captura do evento. Este método garante ordenação precisa independentemente de ajustes no relógio do sistema. Para eventos de rede, o timestamp é aplicado durante o processamento XDP antes de qualquer outro processamento. Para eventos de sistema, a marcação ocorre no handler do kprobe/tracepoint antes da execução da syscall.

3.2.6. Detector

O Detector implementa um ensemble de modelos de ML para identificação de comportamentos maliciosos. O ensemble combina XGBoost e MLP através de uma arquitetura de classificação em duas fases.

Na primeira fase, o modelo XGBoost atua como filtro rápido, processando um conjunto reduzido de características de alta discriminação. Este modelo é otimizado para alta velocidade de inferência e baixo consumo de recursos, classificando eventos em três categorias: benignos (descartados), suspeitos (encaminhados para análise detalhada) e maliciosos (alertados imediatamente). A implementação utiliza o algoritmo gradient boosting com profundidade máxima de árvore limitada a 6 e número de estimadores fixado em 100, balanceando complexidade e capacidade discriminativa.

Eventos classificados como suspeitos são processados pelo modelo MLP na segunda fase, que utiliza um conjunto expandido de características. O MLP é configurado com três camadas ocultas (256, 128, 64 neurônios) e função de ativação ReLU, seguido por uma camada de saída com ativação softmax para classificação multi-classe. Esta arquitetura permite capturar relações mais complexas entre características e identificar padrões sutis de atividade maliciosa.

Os resultados destes modelos são combinados através de um mecanismo de votação ponderada, onde os pesos são determinados pelo grau de confiança de cada modelo e ajustados continuamente com base no feedback de análise de falsos positivos/negativos. Uma camada adicional de análise baseada em regras avalia os resultados do ensemble para incorporar conhecimento de domínio específico e reduzir falsos positivos em cenários conhecidos.

3.3. Otimizações Específicas

O CrossLayerGuardian implementa diversas otimizações para maximizar performance. Uma das principais é o uso de perfect hashing para acesso eficiente a mapas eBPF críticos. Esta técnica implementa uma função hash perfeita para tuplas de conexão, calculada através do algoritmo CMPH (Cache-efficient Minimal Perfect Hashing) modificado para execução em contexto eBPF. Esta implementação elimina colisões e reduz o tempo de busca de O(log n) para O(1), resultando em aceleração significativa para operações de alta frequência como classificação de pacotes.

As estruturas lock-free são utilizadas extensivamente para minimizar contenção em operações concorrentes. Particularmente, o sistema implementa uma versão modificada de ring buffer que utiliza operações atômicas e fence instructions para garantir consistência sem bloqueios, permitindo processamento paralelo eficiente em sistemas multi-core. Esta implementação mantém contadores separados para leitura e escrita com operações atômicas de compare-and-swap para atualização de estado.

O sistema emprega ainda compressão de dados consciente de contexto, onde eventos frequentes e previsíveis são codificados em formatos compactos customizados em vez de estruturas genéricas. Por exemplo, operações comuns de leitura de arquivo são representadas em apenas 16 bytes através de codificação diferencial que armazenar apenas diferenças em relação a templates pré-definidos. Esta técnica reduz o volume de dados transferidos entre kernel e userspace em aproximadamente 60% comparado à representação não otimizada.

O mecanismo de cache de dois níveis implementa um cache L1 no kernel para decisões de pré-filtragem e um cache L2 em userspace para resultados intermediários de processamento. O cache L1 utiliza um mapa LRU (Least Recently Used) para armazenar resultados de decisões recentes, evitando recomputação para padrões recorrentes. O cache L2 utiliza uma estrutura adaptativa que prioriza o armazenamento de resultados para padrões frequentemente observados mas de computação intensiva.

3.4. Mecanismo de Correlação Cross-Layer

A correlação cross-layer é implementada através de três elementos fundamentais: sincronização temporal, mapeamento processo-conexão e buffer de correlação adaptativo.

A sincronização temporal garante ordenação precisa de eventos, utilizando timestamps de kernel e compensação de latência diferencial. O mapeamento processo-conexão rastreia relacionamentos entre processos locais e suas conexões de rede associadas, permitindo vincular atividades observadas nos diferentes domínios.

O buffer de correlação adaptativo mantém contexto histórico necessário para identificar padrões temporais de ataque. Esta implementação utiliza estruturas eficientes para armazenamento de eventos recentes, com estratégias de expiração baseadas em relevância contextual e mecanismos de priorização para eventos potencialmente maliciosos.

3.5. Tolerância a Falhas

O CrossLayerGuardian implementa mecanismos de tolerância a falhas para operação contínua. O subsistema de recuperação automática monitora componentes e reinicia seletivamente aqueles que apresentam falhas, isolando problemas sem comprometer todo o sistema. Os buffers circulares asseguram armazenamento temporário de eventos durante picos de carga ou indisponibilidade temporária de componentes. O checkpoint periódico salva o estado do sistema, incluindo modelos ML, estatísticas de baseline e contextos críticos, permitindo restauração rápida após falhas completas ou atualizações.

3.6. Técnicas de Instrumentação e Estruturas de Dados Especializadas

A implementação combina diferentes técnicas de instrumentação eBPF para maximizar cobertura com mínimo overhead. Para tracepoints, utilizamos BPF trampolines (fentry/fexit) em vez de kprobes quando disponíveis, proporcionando melhoria de performance devido à eliminação da manipulação de registradores. A instrumentação do filesystem segue uma estratégia hierárquica com monitoramento de VFS para captura ampla, filtros específicos por sistema de arquivos para informações detalhadas, e hooking LSM para decisões de acesso sensíveis.

Para instrumentação de rede, a implementação utiliza uma combinação de XDP e TC (Traffic Control) eBPF. O XDP é posicionado para filtragem inicial de alta performance, enquanto o TC permite monitoramento mais detalhado incluindo visibilidade de conexões já estabelecidas.

Tabela 2. Comparação das Estruturas de Dados Implementadas (Compacta)

Estrutura	Uso	Complexidade	Vantagens
LRU com Aging	Rastreamento de conexões	O(1) lookup	Expiração controlada
Bloom Filter	Filtragem rápida	O(k) hashes	Falsos negativos zero
Count-Min Sketch	Contagem de eventos	O(1) update	Espaço constante
Trie compacto	Caminho de arquivos	O(l) (comprimento)	Compressão de prefixo

O sistema utiliza estruturas de dados especializadas como filtros Bloom customizados para detecção eficiente de padrões conhecidos e Count-Min Sketch para contagem eficiente de eventos de alta cardinalidade. Para monitoramento de sistema de arquivos, emprega-se um trie compacto com compressão de prefixos comuns, reduzindo significativamente o consumo de memória em mapas eBPF.

3.7. Pipeline de Machine Learning

O pipeline de machine learning foi implementado com foco em performance de inferência e facilidade de atualização. O módulo de pré-processamento implementa normalização e transformação de features em C++, utilizando instruções SIMD onde disponível para processamento paralelo. Os modelos são serializados em formato binário otimizado com metadados de versionamento, suportando atualização parcial de componentes do ensemble sem reinstalação completa.

Para maximizar throughput, o sistema implementa inferência em batch com tamanho adaptativo, acumulando eventos até um limiar de latência máxima (configurável, padrão 50ms) ou tamanho de batch máximo. Em sistemas com unidades SIMD avançadas ou aceleradores, a implementação detecta e utiliza automaticamente instruções específicas da plataforma.

4. Avaliação

Esta seção apresenta a metodologia de avaliação e resultados experimentais do CrossLayerGuardian, focando em eficácia de detecção, performance e utilização de recursos.

4.1. Ambiente Experimental

Os experimentos foram conduzidos em instâncias AWS EC2 t3.xlarge (4 vCPUs, 16GB RAM) executando Amazon Linux 2023 (kernel 6.1). A topologia de teste incluiu uma instância principal para o CrossLayerGuardian, duas instâncias t3.large para geração de tráfego e uma instância adicional para coleta de métricas. Todas as instâncias foram implantadas na mesma VPC e zona de disponibilidade para minimizar latência de rede.

4.2. Metodologia e Datasets

A avaliação utilizou três fontes de dados complementares:

1. Dataset CICIDS2018: dataset estabelecido para avaliação de IDS, com 80 features por fluxo e proporção 80/20 de tráfego normal/ataques.
2. Tráfego de produção real: capturado de uma aplicação web corporativa durante uma semana (aproximadamente 20GB/dia).
3. Ataques controlados: suite de cinco cenários multi-vetor, incluindo SQL Injection + Shell Upload, Credential Stuffing + Lateral Movement, DDoS + Port Scan, Cryptomining + C2 Communication, e Data Exfiltration + Log Tampering.

Os experimentos focaram em três aspectos: performance base (throughput com tráfego sintético), eficácia de detecção (usando datasets e ataques controlados) e testes de carga (análise de comportamento sob tráfego crescente até 1Gbps).

4.3. Resultados e Análise

O CrossLayerGuardian demonstrou alta eficácia em cenários diversos. Para ataques de rede, alcançou 95,3% de detecção com 1,2% de falsos positivos; para ataques em nível de sistema, 95,1% de detecção com 1,0% de falsos positivos; e para ataques multi-vetor, 95,0% de detecção com 1,2% de falsos positivos. A correlação cross-layer aumentou significativamente a precisão, especialmente em ataques sofisticados que combinam atividades de rede e sistema, detectando 43% mais variantes em comparação com soluções NIDS/HIDS isoladas.

Em termos de desempenho, o sistema superou soluções tradicionais, atingindo throughput máximo de 850 Mbps com XDP offload ativado, mantendo latência média de 85 μs no processamento de pacotes. O monitoramento de syscalls registrou apenas 12 μs de latência média, enquanto o sistema de correlação processou 500.000 eventos por segundo com overhead de rastreamento inferior a 5% da memória utilizada. A utilização de CPU permaneceu abaixo de 8% em média (picos de 12%), com consumo de memória entre 4,2 GB (baseline) e 6,8 GB (pico).

Como demonstrado na Tabela 3, o CrossLayerGuardian apresenta vantagens significativas sobre sistemas estabelecidos como Suricata e Falco, combinando melhor throughput, menor consumo de recursos e superior precisão de detecção. Estes resultados validam a eficácia da abordagem cross-layer para identificação de ataques multi-estágio, demonstrando um equilíbrio entre performance, precisão e utilização de recursos que viabiliza sua implementação em ambientes de produção.

Tabela 3. Comparativo de Performance (Compacto)

Sistema	Throughput	CPU%	Detecção	F. Positivos	Latência
CrossLayerGuardian	850 Mbps	8%	95%	1,2%	85 μs
Suricata	425 Mbps	42%	88,2%	2,5%	120 μs
Falco	N/A	32%	84,5%	1,8%	N/A

5. Conclusão

Este trabalho apresentou o CrossLayerGuardian, um sistema híbrido de detecção de intrusão que preenche efetivamente a lacuna entre o monitoramento de segurança de rede e sistema em um único host. Nossa principal contribuição é uma arquitetura unificada que possibilita correlação cross-layer eficiente através de estruturas de dados baseadas em eBPF e processamento sincronizado de eventos. A implementação de alta performance alcança throughput de 850 Mbps com latência sub-milissegundo, enquanto técnicas avançadas de correlação cross-layer melhoram a precisão de detecção para ataques multi-vetor sofisticados. O design eficiente em termos de recursos mantém o overhead de CPU abaixo de 8%, tornando-o adequado para ambientes de produção. Os resultados experimentais demonstram que o CrossLayerGuardian supera as soluções existentes tanto em precisão de detecção e eficiência de processamento dentro dos testes realizados. O sistema aborda com sucesso as limitações das abordagens tradicionais ao possibilitar correlação cross-layer precisa entre eventos de rede e sistema enquanto mantém performance em nível de produção.

Embora o sistema demonstre fortes capacidades de detecção, a implementação atual opera como um IDS. Ele identifica e alerta sobre atividades suspeitas, mas não implementa mecanismos ativos de bloqueio. Uma direção clara para trabalhos futuros é estender o CrossLayerGuardian com mecanismos de resposta automatizada, transformando-o em um sistema de prevenção (IPS) completo, capaz de agir sobre as ameaças detectadas em tempo real. Outros trabalhos futuros incluem estender o sistema para suportar cenários de implantação distribuída e explorar técnicas avançadas de ML para detecção de ataques zero-day. Estas melhorias ampliarão ainda mais as capacidades do CrossLayerGuardian em endereçar desafios emergentes de segurança em ambientes computacionais modernos.

Referências

Business, V. (2023). 2023 data breach investigations report. Technical report, Verizon Business.

Byrnes, J., Smith, K., and Johnson, R. (2023). A modern implementation of system call sequence based host-based intrusion detection systems. In *International Conference on Network and System Security*, pages 145–160.

Chen, Z., Simsek, M., Kantarci, B., Bagheri, M., and Djukic, P. (2023). Host-based intrusion detection approaches. arXiv:2306.09451.

eBPF Community (2021). ebpf - tutorials & community resources.

Findlay, W. (2020). Host-based anomaly detection with extended bpf.

for Cybersecurity, E. U. A. (2023). Enisa threat landscape 2023. Technical report, ENISA.

for Internet Security, C. (2023). Global threat report 2023: Multi-vector attacks on the rise. Technical report, Center for Internet Security.

Hadi, F., Xiao, B., and Wang, H. (2023). Kernel-level intrusion detection mechanisms. *IEEE Transactions on Dependable and Secure Computing*, 20:1678–1691.

Khraisat, A., Gondal, I., Vamplew, P., and Kamruzzaman, J. (2019). Survey of intrusion detection systems. *Cybersecurity*, 2.

Kostopoulos, S. (2023). Machine learning-based near real time intrusion detection and prevention system using ebpf.

Madhavi, M. and Nethravathi, N. (2023). Intrusion detection systems using advanced machine learning. In *International Conference on Advances in Electronics, Communication, Computing and Intelligent Information Systems*, pages 139–145.

Quincozes, S., Albuquerque, C., Passos, D., and Mossé, D. (2021). Survey of ebpf security mechanisms. *Computer Networks*, 184:107679.

Scikit-learn (2014). Multi-layer perceptron.

Security, I. (2023). Cost of a data breach report 2023. Technical report, IBM Corporation.

Song, S., Suneja, S., Le, M., and Tak, B. (2023). Value of ebpf in cloud computing security. In *IEEE International Conference on Cloud Computing*, pages 296–307.

Vaishali, R. (2023). Hybrid intrusion detection approaches. In *7th International Conference on Computing Methodologies and Communication*, pages 1106–1111.

Wang, G. and Chang, R. (2023). Design and implementation of an intrusion detection system using extended bpf in the linux kernel. In *IEEE/IFIP Network Operations and Management Symposium*, pages 1–9.

Wang, X. and Lu, X. (2020). Host-based intrusion detection mechanisms. *Wireless Communications and Mobile Computing*, 2020:1–13.

XGBoost (2022). Introduction to boosted trees.

Zhang, C., Jia, D., Wang, L., Wang, W., Liu, F., and Yang, A. (2022). Comparative analysis of intrusion detection techniques. *Computers & Security*, 121:102861.

Zhang, L., Wu, H., and Lu, W. (2023). Real-time network intrusion detection system. *IEEE Transactions on Network Service Management*, 20:1232–1245.