Utilizando Estratégias de Monitoramento Leve em Ambientes Conteinerizados para Detecção de Anomalias via HIDS

Anderson Frasão[1], Tiago Heinrich[2], Vinicius Fulber-Garcia[1], Newton C. Will[3], Rafael R. Obelheiro[4], Carlos A. Maziero[1]

[1] Departamento de Informática
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

[2]Max Planck Institute for Informatics (MPI)
Saarbrücken – Saarland – Germany

[3]Departamento de Ciência da Computação
Universidade Tecnológica Federal do Paraná (UTFPR)
Dois Vizinhos – PR – Brasil

[4]Programa de Pós-Graduação em Computação Aplicada
Universidade do Estado de Santa Catarina (UDESC)
Joinville – SC – Brasil

{aacf20,vinicius,maziero}@inf.ufpr.br, theinric@mpi-inf.mpg.de,
will@utfpr.edu.br, rafael.obelheiro@udesc.br

Abstract. *The increased implementation of container-based virtualized environments has raised security concerns due to their proximity to host systems. In this scenario, strategies using intrusion detection through anomalies have emerged as an option to identify and alert about unexpected behaviors. This paper proposes the use of interactions between container and operating system for anomaly detection, executing lightweight and internal monitoring processes within the containerized environment, generating data and traces for the training and execution of machine learning models aimed at distinguishing normal from anomalous behaviors. Thus, the central discussion of this work revolves around the suitability of the data generated by lightweight monitoring tools, represented by sysdig, in the training of models and their subsequent use in HIDS solutions. This potential was assessed through a series of tests, where models trained with data provided by sysdig achieved significant results. They attained high rates of accuracy, precision, recall, F1-Score, and other indicators in the scenarios considered.*

Resumo. *O aumento da implementação de ambientes virtualizados baseados em contêineres tem gerado preocupações de segurança devido à sua proximidade com os sistemas hospedeiros. Nesse cenário, emergiram estratégias que utilizam a detecção de intrusões por meio de anomalias como uma opção para identificar e alertar sobre comportamentos inesperados. Este trabalho propõe o uso de interações entre contêiner e sistema operacional na detecção de anomalias, executando processos de monitoramento leve e interno ao ambiente conteinerizado, gerando dados e traços para o treinamento e emprego de modelos*

de aprendizado de máquina que visam distinguir comportamentos normais de comportamentos anômalos. Assim, a discussão central deste trabalho versa sobre a adequabilidade dos dados gerados pelas ferramentas de monitoramento leve, representadas pelo sysdig, no treinamento de modelos e subsequente uso dos mesmos em soluções de HIDS. Esse potencial foi avaliado por meio de uma série de testes, nos quais os modelos treinados com dados fornecidos pelo sysdig alcançaram resultados significativos, com altas taxas de acurácia, precisão, recall, F1-Score, além de outros indicadores, nos cenários considerados.

1. Introdução

Ambientes de computação modernos são frequentemente estabelecidos em equipamentos distribuídos e compartilhados, nos quais cotas de recursos são adquiridas sob demanda [Randal 2020]. Esses ambientes dependem intimamente de tecnologias de virtualização para criar espaços isolados em *software* (máquinas virtuais e contêineres), destinados a hospedar e atender a múltiplos serviços e clientes diversos.

Particularmente, a introdução de aplicações em contêineres representa um marco significativo na evolução tecnológica. Essa tecnologia, difundida no ambiente Linux, permite que múltiplas aplicações compartilhem um único sistema operacional, eliminando a necessidade e a sobrecarga de replicação de sistemas operacionais para cada uma delas, uma característica típica das máquinas virtuais. Dessa maneira, os contêineres destacam-se devido à sua rápida instanciação, alta escalabilidade e baixo consumo de recursos computacionais quando comparados às máquinas virtuais tradicionais [Sturm et al. 2017].

Porém, uma vez que os contêineres são executados sobre um sistema operacional hospedeiro comum e compartilhado, estes apresentam isolamento limitado quando comparados às máquinas virtuais [Flauzac et al. 2020]. Assim, vulnerabilidades em contêineres surgem principalmente em dois contextos: interno e externo. As vulnerabilidades internas ao contêiner incluem componentes desatualizados, inclusão de arquivos maliciosos e práticas inadequadas de gerenciamento de dados e usuários. Já as vulnerabilidades de natureza externa aos contêineres são herdadas do sistema hospedeiro, além de potenciais ataques que visam, através desse sistema, invadir um contêiner por ele hospedado.

A prevenção das ameaças relacionadas às vulnerabilidades de contêineres, assim como a mitigação destas, quando se manifestam, demanda, principalmente, a adoção de boas práticas de segurança, como atualizações regulares, controle de privilégios, monitoramento constante e a adoção de políticas de segurança apropriadas. Além disso, a seleção de imagens confiáveis e a verificação de assinaturas digitais antes da instanciação são práticas importantes [Martínez-Magdaleno et al. 2021].

Outros componentes importantes das rotinas de prevenção e mitigação são processos contínuos de monitoramento e detecção precoce de falhas, ataques e vulnerabilidades [Röhling et al. 2019]. Nesse contexto, os Sistemas de Detecção de Intrusão Baseados em *Host* (HIDS) desempenham um papel crítico, permitindo a inspeção das chamadas de sistema (*system calls*) e a análise dos processos executando em um ambiente virtualizado ou hospedeiro. Em especial, destacam-se as soluções de HIDS baseadas em anomalias, que são treinadas a partir do comportamento normal de um sistema, buscando identificar comportamentos anômalos, que podem caracterizar potenciais ameaças.

As ferramentas de monitoramento e coleta de dados e traços necessários para projetar, implementar e executar soluções de HIDS podem ter maiores ou menores impactos no hospedeiro e nos contêineres monitorados em um ambiente virtualizado. Algumas soluções utilizadas nesse contexto, como as baseadas em *strace* [Levin et al. 2024], podem gerar interrupções frequentes na execução dos processos monitorados, causando impactos significativos na função ou serviço executado. Outras ferramentas, como *sysdig* [Platform 2024], realizam monitoramento ao nível de contêiner e são menos intrusivas aos processos executados nos contêineres, não causando grandes efeitos colaterais.

Este trabalho explora e propõe estratégias para a detecção de intrusão baseada em anomalias em um ambiente de virtualização ao nível de contêiner, utilizando a ferramenta *sysdig*. Argumenta-se que processos de monitoramento que não sejam prejudicados pelo ruído de uma *container engine* apresentam características favoráveis para a detecção de anomalias, sem comprometimento do isolamento das aplicações em execução nas instâncias dos contêineres. Por meio dos dados e traços obtidos nesse processo de monitoramento, modelos para a classificação de anomalias são treinados e testados como soluções de HIDS.

Como caso de estudo, usamos uma aplicação *Wordpress* em um contêiner Docker, visando monitorar seu comportamento tanto em situações normais de uso quanto em cenários de ataque ou de execução de operações maliciosas. Por fim, as soluções de HIDS geradas a partir dos traços coletados foram testadas no mesmo ambiente, objetivando avaliar a eficiência e a eficácia da detecção de anomalias a partir destas.

Pontualmente, as contribuições do trabalho consistem em:

- Avaliar um método de monitoramento ao nível de contêiner, verificando o potencial de dados coletados pelo *sysdig* para a implementação de soluções de HIDS;
- Apresentar novas estratégias de detecção de anomalias em um ambiente virtualizado habilitado por contêineres.

O restante do artigo está organizado como segue: a Seção 2 apresenta os principais conceitos necessários para a compreensão da proposta. A Seção 3 revisa os trabalhos relacionados. A Seção 4 detalha, tecnicamente, a proposta. A Seção 5 apresenta e discute os testes e resultados obtidos, e a Seção 6 conclui o trabalho.

2. Fundamentação Teórica

Esta seção apresenta a fundamentação teórica para a compreensão do trabalho, incluindo sistemas de virtualização baseados em contêiner (Seção 2.1), detecção por anomalias (Seção 2.2) e uso de chamadas de sistema em segurança (Seção 2.3).

2.1. Virtualização com Contêineres

A virtualização baseada em contêineres utiliza o *kernel* de um sistema para isolar processos, diferindo da virtualização com hipervisor, pois não possuem seu próprio *hardware* virtualizado. Contêineres usam o *hardware* do sistema hospedeiro, comunicando-se diretamente com seu *kernel*. Essa abordagem permite inicializações rápidas, em milissegundos, e maior eficiência em comparação às máquinas virtuais convencionais. As imagens de contêineres são geralmente menores, pois não necessitam incluir uma cadeia completa

de ferramentas para executar um sistema operacional. Essa eficiência resulta da eliminação da emulação de *hardware* e da inicialização de um sistema operacional completo [Eder 2016].

Na Figura 1 é ilustrada a comparação entre arquiteturas de virtualização baseada em contêiner e a virtualização baseada em hipervisor. A virtualização baseada em contêineres difere da abordagem de máquinas virtuais completas, criando instâncias isoladas do espaço do usuário e compartilhando um único *kernel* do sistema operacional entre elas [Xavier et al. 2013]. Exemplos incluem o Linux-VServer, que introduz recursos no *kernel* para isolamento, e o Docker, uma ferramenta de linha de comando facilitadora na criação e gerenciamento de contêineres, maximizando o uso da infraestrutura [Eder 2016, Hat 2024]. O Kubernetes, uma plataforma de orquestração de contêineres de código aberto, oferece recursos avançados como escalonamento automático e gerenciamento de armazenamento [Kubernetes 2024].

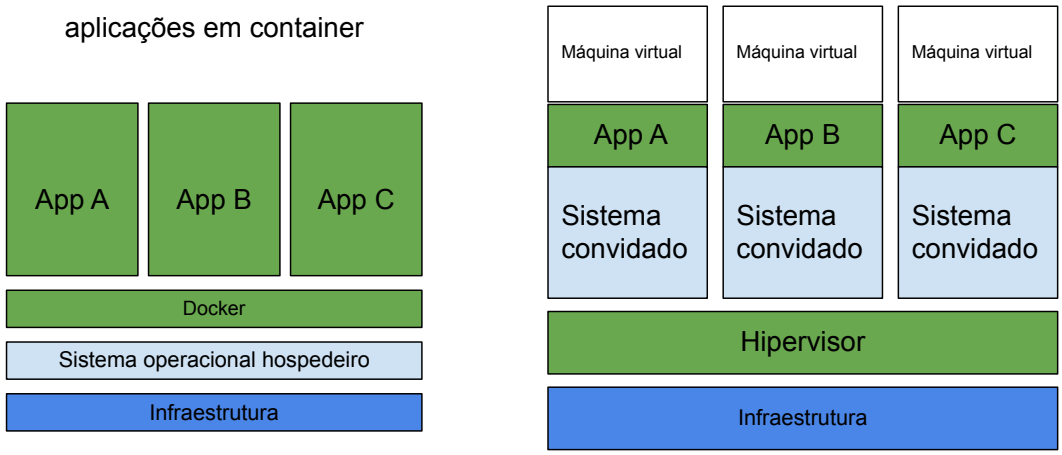

Figura 1. Comparação entre VM e Contêiner, com base em [Docker 2024].

A tecnologia de virtualização de nível de sistema operacional, conhecida como contêiner Linux, isola e agrupa processos em um ambiente que compartilha o *kernel* Linux entre os contêineres. Para isso, são utilizados os mecanismos de *namespace* e *cgroups*, que visam fornecer isolamento e limitar o consumo de recursos computacionais, como CPU e memória. No entanto, a interdependência entre esses mecanismos pode impactar aspectos de segurança dos contêineres [Lin et al. 2018].

Particularmente, os *namespaces* do *kernel* criam espaços de nomes separados para processos, facilitando a migração de contêineres. Já os *cgroups* rastreiam e controlam processos e grupos, oferecendo flexibilidade na gestão de recursos. A aplicação de *Mandatory Access Control* (MAC) fortalece os mecanismos de controle de acesso no Linux/Unix. Essas ferramentas são cruciais para a virtualização de contêineres, proporcionando isolamento, portabilidade, eficiência e escalabilidade, enquanto adicionam uma camada de proteção contra ataques internos.

Os ambientes de contêiner oferecem uma variedade de recursos, desde a criação de imagens personalizadas até a orquestração eficiente em *clusters*, visando otimizar a produtividade e a confiabilidade no ciclo de vida das aplicações. O ambiente mais popular é o Docker, uma plataforma de código aberto [Hat 2024]. Sua abordagem modular

permite desativar partes de uma aplicação para reparo sem interromper sua execução completa, seguindo uma abordagem semelhante à arquitetura orientada a serviços. Utilizando camadas e controle de versão de imagens, o Docker otimiza o processo de criação de contêineres, possibilitando a reutilização e rápida implantação. A função de reversão permite recuperar versões anteriores devido às camadas criadas, alinhando-se com práticas ágeis de desenvolvimento. A implantação rápida é alcançada em segundos, compartilhando contêineres facilmente e eliminando a necessidade de reiniciar o sistema operacional, resultando em eficiência no desenvolvimento da aplicação e economia na criação e destruição de dados associados aos contêineres.

2.2. Detecção de Anomalias

Detecção de anomalias é uma estratégia para a detecção de padrões em dados que não seguem um comportamento normal e podem ser causados por atividades maliciosas ou falhas no sistema [Chandola et al. 2009]. Sistemas de detecção de intrusões baseados em anomalias geralmente coletam eventos de interesse que ocorrem em um sistema alvo e constroem um modelo estatístico que descreve esses dados durante a operação normal do sistema. Posteriormente, dados coletados em um sistema em operação são avaliados tomando esse modelo como base, e eventuais desvios do comportamento esperado são reportados como anomalias [Jyothsna and Rama Prasad 2011].

A detecção de anomalias é uma atividade crucial na análise de dados, visando identificar padrões raros que podem indicar eventos significativos em diversos domínios de aplicação [Ahmed et al. 2016]. As principais técnicas para detecção de anomalias abrangem métodos baseados em distância, densidade, modelos estatísticos, aprendizado de máquina e mineração de dados. Métodos de distância, como *k-Nearest Neighbor* e *Local Outlier Factor* [Huang and Zhang 2019], identificam pontos distantes da maioria. Métodos de densidade, como DBSCAN e OPTICS, identificam pontos com densidade menor que os vizinhos. Métodos estatísticos, como *Boxplot* e método de Dixon, modelam a distribuição dos dados [Barbato et al. 2011]. Abordagens de aprendizado de máquina, como *Isolation Forest* e *One-class SVM*, identificam padrões que diferem do comportamento aprendido. Mineração de dados, com algoritmos como ROD e COP, identifica padrões anormais usando técnicas de associação de regras e agrupamento [Hodge and Austin 2004].

2.3. Uso de Chamadas de Sistema em Segurança

As chamadas de sistema representam a interface que um sistema operacional oferece às aplicações [Tanenbaum and Bos 2016]. Elas permitem, entre outras funcionalidades, alocar e desalocar recursos, realizar entrada e saída (incluindo manipulação de arquivos), efetuar comunicação e sincronização entre processos, e criar e destruir processos/*threads*.

Como as chamadas de sistema mediam o acesso de processos de usuário a recursos críticos do sistema, elas constituem um ponto privilegiado de observação e controle do comportamento dessas aplicações do ponto de vista de segurança [Forrest et al. 1996]. Apesar de existirem diversas abordagens para inserção de códigos alterados em uma máquina, os ataques compartilham a característica de explorar a interface de chamada do sistema para realizar ações mal-intencionadas, como gravar no disco ou enviar pacotes de rede. O monitoramento de chamadas do sistema é uma técnica amplamente empregada para detectar aplicações comprometidas, e possibilita também encerrá-las ou colocá-las

em ambiente controlado para mitigar danos. Essa abordagem baseia-se em uma política que captura as chamadas de sistema realizadas por uma aplicação e extrai um perfil que é comparado a um perfil de comportamento normal, interrompendo a execução se desvios forem detectados. Embora o monitoramento de chamadas do sistema não ofereça uma proteção completa, é uma ferramenta valiosa quando combinada com outras técnicas para dificultar invasões e minimizar impactos [Rajagopalan et al. 2006].

3. Trabalhos Relacionados

A utilização de contêineres na computação em nuvem proporciona diversas vantagens em termos de eficiência e escalabilidade. No entanto, traz consigo desafios significativos no que diz respeito à segurança. O trabalho de [Sultan et al. 2019] aborda de maneira abrangente os problemas e desafios de segurança associados ao emprego de contêineres nesse contexto. Entre as principais preocupações, destacam-se a segurança do host, o gerenciamento de identidade e acesso, a proteção do registro de contêineres e a segurança intrínseca dos próprios contêineres.

Diversas propostas abordam estratégias de detecção de anomalias e intrusões em ambientes de contêiner. Trabalhos relevantes relacionados a essas propostas são sumarizadas a seguir.

[Abed et al. 2015] apresenta um sistema de detecção de intrusão baseado em contêineres para aplicações Linux, utilizando um modelo de perfil de comportamento do aplicativo. [Du et al. 2018] propõe uma abordagem para detectar anomalias em microsserviços baseados em contêineres, utilizando análise de séries temporais e aprendizado de máquina. [Zou et al. 2019] introduz um sistema de monitoramento de anomalias para contêineres Docker com base na técnica de *Isolation Forest* otimizada.

[Flora and Antunes 2019] avalia a eficácia de diferentes abordagens de detecção de intrusão em ambientes de contêineres *multi-tenant*, concluindo que a detecção baseada em comportamento é eficaz. [Srinivasan et al. 2019] propõe uma solução de detecção de intrusões em tempo real em contêineres Docker, utilizando um modelo probabilístico baseado em redes bayesianas.

[Castanhel et al. 2020] investiga a detecção de anomalias em ambientes de contêineres, explorando o impacto da visão parcial das informações nos resultados. Seu estudo envolve o uso de *Isolation Forest* e *One-Class SVM*, considerando dados brutos e filtrados. Em um trabalho subsequente, [Castanhel et al. 2021] foca em uma aplicação específica em um ambiente de contêiner, explorando técnicas de aprendizado de máquina para identificar ataques.

[Rocha et al. 2022] propõe um *framework* para a implementação de HIDS baseado em anomalias de chamadas de sistema em ambientes de contêineres. [Shen et al. 2022] aborda a vulnerabilidade de contêineres a ataques de segurança, propondo uma nova estrutura de detecção de anomalias usando um algoritmo de *cluster*.

Por fim, os estudos revisados apresentam sistemas baseados em chamadas de sistema, perfil de comportamento e monitoramento de desempenho. A eficácia de abordagens de detecção, como as baseadas em rede, comportamento e assinaturas, é discutida, enfatizando o papel do aprendizado de máquina na melhoria da detecção em ambientes multi-inquilinos. Apesar das contribuições valiosas, são identificadas limitações nos

trabalhos, como eficiência insatisfatória, desafios na implementação prática e indisponibilidade dos *datasets* utilizados.

4. Proposta

Este trabalho objetiva apresentar estratégias de detecção de anomalias e intrusões em contêineres Docker. Para isso, considera-se uma abordagem de isolamento da aplicação e coleta de informações em um nível de contêiner, levando em conta características das instâncias Docker, como o isolamento de recursos e uso de *namespaces*. A ferramenta *sysdig* foi empregada para a execução de tal tarefa. Utilizando as informações e traços de execução monitorados e coletados, objetiva-se investigar o potencial do uso das informações extraídas pela ferramenta no processo de detecção de intrusão.

A ferramenta *sysdig* foi utilizada, pois esta se mostra adequada para a captura de métricas do sistema, como CPU, memória e E/S de disco, além de informações adicionais relacionadas a eventos e chamadas do *kernel* [Platform 2024]. O arquivo de saída de captura inclui identificadores de eventos, *timestamp*, *thread*, nome e ID do contêiner, nome e ID do processo de aplicação, nome da chamada do sistema e parâmetros. No processo de utilização dos dados, apenas o nome das chamadas do sistema é mantido, considerando apenas o número de vezes que uma chamada é feita nos testes. O *sysdig* também possui vantagens ao considerar o desempenho da estratégia de coleta, já que um *buffer* é utilizado para armazenar os dados, diminuindo a sobrecarga gerada na aplicação de outras ferramentas.

Uma das dessas outras ferramentas amplamente conhecida e utilizada para coleta de informações por chamadas de sistema é o *strace*. Ele utiliza a chamada de sistema *ptrace*, fornecida pelo Linux e outros sistemas operacionais, para instrumentar um processo-alvo e monitorar suas chamadas de sistema. Quando uma chamada de sistema é invocada, o *strace* interrompe o processo rastreado, captura a chamada, decodifica-a e, em seguida, retoma a execução do processo. No entanto, essa abordagem possui uma desvantagem significativa: cada mudança de contexto original é transformada em várias mudanças de contexto de monitoramento, reduzindo o desempenho geral da aplicação e mantendo o processo rastreado congelado até que o *strace* realize a decodificação [Platform 2024].

Por outro lado, o *sysdig* [Platform 2024] adota uma arquitetura diferente. Ele utiliza um *driver* chamado *sysdig-probe*, que captura eventos no *kernel* por meio de uma funcionalidade chamada *tracepoints*. Esses *tracepoints* permitem que um "*handler*" seja instalado e chamado em funções específicas do *kernel*. O *handler* do *sysdig-probe* é simples e limitado a copiar os detalhes do evento em um *buffer* compartilhado, codificado para consumo posterior. Os eventos são então mapeados em espaço de usuário, permitindo acesso direto ao *buffer* sem cópias adicionais, minimizando o uso da CPU e as perdas de cache. Duas bibliotecas, *libscap* e *libsinsp*, fornecem suporte para leitura, decodificação e análise desses eventos. O *sysdig* age como uma camada de abstração simples em torno dessas bibliotecas.

Assim, após a execução com monitoramento via *sysdig*, os traços coletados pelo mesmo foram processados para extrair características relevantes e reduzir o ruído, considerando apenas informações potencialmente significativas no processo de detecção. Desta forma, para cada amostra benigna e maliciosa, 10 execuções foram realizadas.

O conjunto de dados resultante foi empregado no treinamento dos modelos de aprendizado de máquina. Para o treinamento, optou-se por usar o *grid search* para determinar os melhores parâmetros para os respectivos modelos, bem como investigar a qualidade dos traços para um processo de detecção de anomalias. Com base no modelo treinado, este é usado como um HIDS capaz de detectar anomalias e intrusões em tempo real, considerando qualquer desvio significativo em relação aos padrões estabelecidos como uma possível ameaça.

O processo de desenvolvimento dos HIDS baseados em modelos de aprendizado de máquina é resumido em quatro etapas:

1. **Coleta de dados:** Utilização da ferramenta *sysdig* para capturar chamadas de sistema e informações relevantes sobre as atividades em execução dentro do contêiner Docker.
2. **Pré-processamento:** Os dados coletados são pré-processados para extrair características relevantes e reduzir o ruído, garantindo a consideração apenas de informações significativas no processo de detecção.
3. **Treinamento:** Aplicação de técnicas de aprendizado de máquina para treinar o HIDS em um conjunto de dados anotados, identificando padrões normais de comportamento do contêiner.
4. **Detecção de anomalias:** Com base no modelo treinado, o HIDS é testado na detecção de anomalias e intrusões em tempo real, considerando qualquer desvio significativo em relação aos padrões estabelecidos como uma possível ameaça.

Particularmente, cabe ressaltar que uma operação de coleta do *sysdig* contém os seguintes elementos: um identificador de evento produzido pela ferramenta, um *timestamp* da chamada, a *thread* que está processando a solicitação, o nome e ID do contêiner, o nome e ID do processo do aplicativo que está se comunicando com o contêiner/kernel, e o nome da *system call* com seus parâmetros/argumentos.

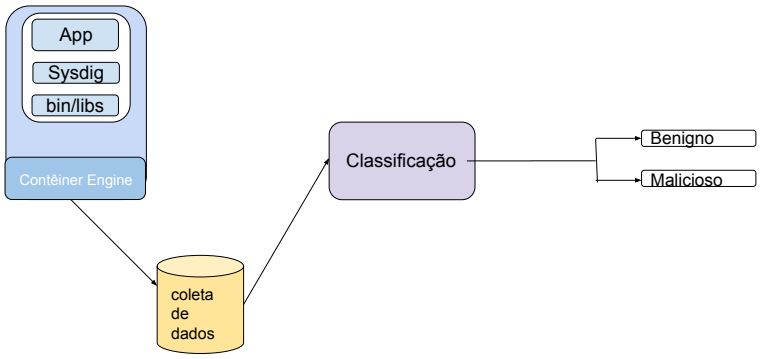

Figura 2. Estrutura para a coleta de dados e detecção de anomalias.

Com o intuito de avaliar uma estratégia viável para detectar anomalias com base em chamadas de sistema, a abordagem proposta concentra-se em uma aplicação em execução em um contêiner. O objetivo é investigar o uso de técnicas de aprendizado de máquina para identificar ataques e compreender o comportamento dos sistemas de detecção de intrusão sob essa perspectiva, utilizando um HIDS. O modelo de execução do

HIDS gerado como resultado está representado, em alto nível, na Figura 2. A proposta visa discutir *(i)* o potencial do uso de chamadas de sistema coletadas pelo *sysdig* para a detecção de anomalias e *(ii)* a viabilidade de modelos de aprendizado de máquina serem usados como HIDS no processo de detecção de anomalias.

5. Avaliação Experimental

Esta seção apresenta os experimentos e a avaliação dos resultados, abrangendo a organização do ambiente de testes (Seção 5.1), a descrição do conjunto de dados utilizados (Seção 5.2) e a discussão dos resultados observados (Seção 5.3).

5.1. Descrição do Cenário

Os experimentos foram realizados em um ambiente Linux 5.15.0-56-generic 62-Ubuntu utilizando a distribuição Linux Mint 21.2, o ambiente de virtualização escolhido para a realização dos testes foi o Docker na versão 20.10.21.

Para a experimentação, foram selecionados sete algoritmos de aprendizado de máquina [Sarker 2021]: *Random Forest* (RF), *XGBoost* (XGB), *Decision Tree* (DT), *Nu-Support Vector* (NuSV), *Multi-layer Perceptron* (MLP), *AdaBoost* (AB) e *Stochastic Gradient Descent* (SGD). Esses modelos visam ilustrar o comportamento do processo de aprendizagem ao avaliar as características escolhidas para as categorias de classificação "benigno"e "malicioso", sendo amplamente utilizados em soluções de detecção de anomalias na literatura [Mishra et al. 2018, Ceschin et al. 2024, Heinrich et al. 2024].

Para cada algoritmo de aprendizado de máquina, um modelo foi treinado com os dados coletados pela ferramenta *sysdig*, sendo empregado como um HIDS. Os experimentos visam aferir a eficácia no uso de chamadas de sistema para a identificação de anomalias.

5.2. Coleta e Conjunto de Dados

O processo de coleta de dados ocorreu utilizando a estrutura descrita na Figura 2, sendo o *Wordpress*, versão 4.9.2, executado em um contêiner o alvo da coleta. Para os testes, a simulação dos comportamentos normais e anormais ocorreu de forma distinta: (i) para a simulação de comportamento normal, foram elaboradas 10 rotinas de execução envolvendo atividades corriqueiras de interação com uma página de *blog*, e cada rotina foi executada 10 vezes, onde o fluxo de chamadas de cada execução foi salvo em um arquivo diferente, totalizando 100 arquivos com fluxos de comportamento normal.

Já a simulação do comportamento anômalo ocorreu de forma similar à do comportamento benigno, mas explorando dez vulnerabilidades diferentes:

- CVE-2019-9978 — Plugin Social Warfare (versão < 3.5.3). Permite a execução de código arbitrário por meio de uma funcionalidade que gerencia a importação de configurações;
- CVE-2020-25213 — Plugin File Manager (wp-file-manager) (versão <= 6.9). Permite o *upload* e execução de código PHP arbitrário;
- Vulnerabilidade no plugin Simple File List (simple-file-list) (versão < 4.2.3). Permite o *upload* de código arbitrário e sua execução no contexto do processo do servidor da web, o que pode facilitar acesso não autorizado ou aumento de privilégios;

- Vulnerabilidade no plugin Payments Forms (versão 2.4.6). Permite a injeção de código arbitrário;
- CVE-2022-3142 — Plugin NEX-Forms (versão < 7.9.7). Permite a injeção de SQL autenticada;
- Vulnerabilidade no plugin Mail Masta (versão 1.0). Permite inclusão de arquivos, geralmente explorando um mecanismo de "inclusão dinâmica de arquivos" implementado no aplicativo de destino;
- Vulnerabilidade no plugin Really Simple Guest Post (versão 1.0.6). Permite o *upload* de arquivos indevidos;
- CVE-2015-5065 — Plugin Paypal Currency Converter Basic For WooCommerce - File Read (versão < 1.4). Permite o download remoto de arquivos;
- Vulnerabilidade no plugin LeagueManager (versão 3.9.11). Permite a injeção de código SQL;
- Vulnerabilidade no plugin CodeArt Google MP3 Player. Permite o download do arquivo de configuração config.php e a extração de dados de acesso do banco de dados.

Cada rotina de coleta de comportamentos anômalos foi executada 10 vezes, e em cada execução, o fluxo de chamadas (traço de execução) foi direcionado para um arquivo diferente. O resultado foi um total de 100 arquivos que compõem o conjunto de dados anômalos[1].

5.3. Resultados e Discussão

Diversos modelos derivados dos algoritmos de aprendizado de máquina foram avaliados utilizando o *grid search*. Essa abordagem permitiu testar um conjunto variado de parâmetros para determinar os mais apropriados em termos de eficácia dos modelos. Para realizar essa avaliação, foi utilizado o pacote *scikit-learn* [Pedregosa et al. 2011].

Os experimentos realizados com o *grid search* utilizaram um *k-fold* de 5, dividindo o conjunto de dados em cinco partes iguais para treinamento e teste. O desempenho dos modelos foi avaliado considerando as seguintes métricas: *Receiver Operating Characteristic* (ROC) , *Precision* e *Recall* , *F1-Score* (F1S) , *Accuracy* , além de *Balanced Accuracy* (BAC) e *Brier Score* (BS) .

A Tabela 1 apresenta os resultados obtidos após a busca exaustiva dos melhores parâmetros. Os resultados gerais dos classificadores indicam que o uso do *sysdig* para observar o comportamento de aplicações em ambientes virtualizados é promissor.

O modelo *AdaBoost* destacou-se com os melhores resultados de eficácia dentre os métodos avaliados. Com uma precisão de 93,48% e um *recall* de 87,76%, o *AdaBoost* mostrou-se eficaz na identificação de instâncias positivas, tornando-o uma opção promissora para implementação. É importante notar que a alta precisão indica que o modelo possui uma baixa taxa de falsos positivos, o que é crucial em cenários onde identificar incorretamente uma instância positiva pode ter graves consequências. Além disso, o ROC de 97,68% e o BS de 9% demonstram uma excelente capacidade de discriminação e calibração. A sólida taxa de *recall* de 87,76% também sugere que o modelo pode minimizar a ocorrência de falsos negativos, ou seja, a perda de instâncias positivas.

[1]Os dados coletados, tanto benignos, quanto anômalos, estão disponíveis em `https://github.com/Carmofrasao/sbseg24-sysdig`.

Tabela 1. Desempenho dos algoritmos de ML considerando todas as chamadas

Classificador	**ROC**	***Precision***	***Recall***	**F1S**	***Accuracy***	**BAC**	**BS**
AdaBoost	97.68%	93.48%	87.76%	90.53%	91.53%	90.94%	9%
Decision Tree	81.35%	91.67%	67.35%	77.65%	81.00%	80.73%	19%
Multilayer Perceptron	94.00%	95.00%	77.55%	85.39%	87.00%	86.81%	13%
Nu-Support Vector	88.88%	80.07%	93.88%	86.79%	86.00%	86.15%	14%
Random Forest	96.14%	85.71%	85.71%	85.71%	86.00%	85.99%	14%
Stoc. Gradient Descent	80.31%	83.33%	71.43%	76.92%	79.00%	78.85%	21%
XGBoost	96.68%	81.82%	91.84%	86.54%	86.00%	86.11%	14%

Os métodos *Random Forest*, *Nu-Support Vector Classification*, *Multilayer Perceptron* e *XGBoost* apresentaram desempenho médio entre os algoritmos avaliados. Eles demonstraram taxas de precisão e *recall* acima de 77%, indicando uma capacidade confiável de classificar instâncias positivas e negativas. Isso significa que esses modelos mantêm um equilíbrio entre falsos positivos e falsos negativos, o que é importante em muitas aplicações. Os valores de ROC, BAC e BS também estão na média, tornando-os adequados para várias situações. No entanto, é essencial considerar o impacto de falsos positivos e falsos negativos em cada contexto de aplicação específico.

A *Decision Tree* e o *Stochastic Gradient Descent* exibem os piores valores em comparação com os outros métodos avaliados. Embora apresentem taxas de precisão acima de 80%, suas taxas de *recall* são de 67,35% e 71,43%, respectivamente. Isso significa que a DT e o SGD tendem a gerar mais falsos negativos, o que pode ser crítico em situações onde a identificação precisa de instâncias positivas é fundamental. Além disso, seus valores de ROC e BAC são inferiores aos dos demais modelos, indicando uma menor capacidade de discriminação e calibração. O BS mais alto em ambos os classificadores também sugere que eles não são tão bem calibrados quanto os outros métodos, tornando-os menos adequados para aplicações sensíveis a erros.

Ao considerar os valores encontrados para as métricas ROC e F1S, observa-se que os modelos *AdaBoost*, *Random Forest* e *XGBoost* alcançaram os melhores resultados após o processo de busca exaustiva de parâmetros. No entanto, esses modelos ainda são afetados por uma taxa de falsos positivos e falsos negativos, indicando que alguns comportamentos observados pelos modelos no conjunto de dados não foram suficientes para distinguir claramente entre as duas classes de classificação. Esse problema pode ser decorrente do número de amostras utilizadas para o treinamento. Para amenizar esse tipo de classificação incorreta, uma expansão do conjunto de dados pode ser realizada. Esse impacto é observado na precisão, que apresenta um número pequeno de falsos positivos. O *recall* demonstra o impacto de falsos negativos, que induziram os modelos a errar ou subestimar durante o processo de classificação.

Especificamente, os valores alcançados pelo BS evidenciam que os modelos *Multilayer Perceptron* e *AdaBoost* possuem a menor incidência de classificações incorretas. Um padrão similar é observado ao considerar os valores apresentados pelo BAC, que também demonstra a adequação dos modelos. No entanto, destaca-se que os classificadores *Decision Tree* e *Stochastic Gradient Descent* foram fortemente impactados por uma classe que não estava balanceada corretamente.

O *grid search* permitiu a identificação dos parâmetros mais adequados para o processo de detecção de anomalias a partir dos dados fornecidos pelo *sysdig*. A estratégia demonstrou a viabilidade do uso de modelos de aprendizado de máquina na composição de soluções de HIDS a partir dos dados disponíveis. Em particular, os resultados obtidos com o algoritmo *AdaBoost* foram os mais balanceados e adequados em termos de eficácia geral nos cenários de teste considerados.

De maneira geral, **os experimentos cumpriram o seu principal objetivo, de confirmar e evidenciar a adequabilidade do uso dos dados (particularmente, chamadas de sistema) coletados com o *sysdig* para a detecção de anomalias**. Foi possível observar que, a partir dos dados de monitoramento gerados pelo *sysdig*, modelos de aprendizado de máquina eficazes puderam ser gerados. Sendo assim, as características de monitoramento leve do *sysdig* (sem interrupção de processos, por exemplo) podem ser efetivamente aproveitadas no desenvolvimento de HIDS mais eficientes e menos intrusivos para ambientes de virtualização habilitados por contêineres.

6. Conclusão

A adoção de contêineres na evolução tecnológica permite o eficiente compartilhamento de um único sistema operacional entre diversas instâncias de aplicativos. Esse enfoque ganhou destaque pela demanda de implantação rápida de aplicações, oferecendo benefícios como portabilidade, economia de memória e facilidade de migração. Contudo, adotar tal estratégia traz consigo riscos relacionados à implementação, que incluem práticas inadequadas de gerenciamento e a exploração de vulnerabilidades.

Este trabalho analisa a possibilidade e viabilidade da detecção de intrusões por meio de soluções HIDS e processos de monitoramento leve e pouco intrusivo, detectando anomalias em ambientes de virtualização por contêineres. Os dados considerados para a detecção foram provenientes da solução *sysdig* e focaram no conjunto de chamadas de sistema executadas por aplicações em contêineres. Ressalta-se que o monitoramento do *sysdig* não causa interrupções na aplicação, ao contrário de alternativas como o *strace*, conferindo ao primeiro a característica de monitoramento leve.

Para avaliar a viabilidade dos dados coletados para a detecção de anomalias e intrusões, esses dados foram usados para treinar modelos de aprendizado de máquina para HIDS. Todos os modelos foram analisados sob diferentes conjuntos de parâmetros selecionados através do *grid search*, o que possibilitou uma comparação dos melhores modelos desenvolvidos com diferentes algoritmos a partir dos dados fornecidos pelo *sysdig*.

Em especial, o modelo *AdaBoost* se destacou na detecção de anomalias e intrusões, apresentando uma precisão de 93,48% e uma sólida taxa de *recall* de 87,76%, sendo eficaz na identificação de instâncias positivas. Além disso, a ROC medida foi excepcionalmente alta, de 97,68%, e o *Brier Score* foi de 9%, o que demonstra uma capacidade notável de discriminação e calibração do modelo. Esses resultados demonstraram que a execução de um monitoramento leve pode fornecer dados adequados e suficientes para o desenvolvimento de HIDS efetivos.

Trabalhos futuros incluem testar os dados gerados por outras soluções de monitoramento leve (*e.g.*, baseados em eBPF [Jia et al. 2023], Ftrace [Rostedt and Hat 2014] e LTTng [LTTng 2024]), comparando-os com o *sysdig* tanto no processo de monitoramento

quanto na adequabilidade dos dados coletados para a geração de modelos de aprendizagem de máquina, além da eficácia dos modelos ao operarem como HIDS. Além disso, busca-se desenvolver soluções de HIDS mais abrangentes a partir dos dados do *sysdig*; para isso, será necessário coletar e rotular um conjunto de dados rico e extenso, reproduzindo ataques e monitorando-os em ambientes conteinerizados.

Agradecimentos

Este trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), UDESC e FAPESC. Os autores também agradecem o Programa de Pós-Graduação em Informática da UFPR e a UTFPR *Campus* Dois Vizinhos.

Referências

Abed, A. S., Clancy, C., and Levy, D. S. (2015). Intrusion detection system for applications using Linux containers. In *Springer Internationl Workshop on Security and Trust Management*, pages 123–135, Vienna, Austria.

Ahmed, M., Mahmood, A. N., and Hu, J. (2016). A survey of network anomaly detection techniques. *Elsevier Journal of Network and Computer Applications*, 60:19–31.

Barbato, G., Barini, E., Genta, G., and Levi, R. (2011). Features and performance of some outlier detection methods. *Taylor & Francis Journal of Applied Statistics*, 38(10):2133–2149.

Castanhel, G. R., Heinrich, T., Ceschin, F., and Maziero, C. (2021). Taking a peek: An evaluation of anomaly detection using system calls for containers. In *IEEE Symposium on Computers and Communications*, pages 1–6, Athens, Greece.

Castanhel, G. R., Heinrich, T., Ceschin, F., and Maziero, C. A. (2020). Sliding window: The impact of trace size in anomaly detection system for containers through machine learning. In *SBC Escola Regional de Redes de Computadores*, pages 141–146, Porto Alegre, RS, Brazil.

Ceschin, F., Botacin, M., Bifet, A., Pfahringer, B., Oliveira, L. S., Gomes, H. M., and Grégio, A. (2024). Machine learning (in) security: A stream of problems. *ACM Digital Threats: Research and Practice*, 5(1):1–32.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58.

Docker (2024). Use containers to build, share and run your applications. `https://www.docker.com/resources/what-container`.

Du, Q., Xie, T., and He, Y. (2018). Anomaly detection and diagnosis for container-based microservices with performance monitoring. In *Springer International Conference on Algorithms and Architectures for Parallel Processing*, pages 560–572, Guangzhou, China.

Eder, M. (2016). Hypervisor-vs. container-based virtualization. *Future Internet and Innovative Internet Technologies and Mobile Communications*, 1.

Flauzac, O., Mauhourat, F., and Nolot, F. (2020). A review of native container security for running applications. *Procedia Computer Science*, 175:157–164.

Flora, J. and Antunes, N. (2019). Studying the applicability of intrusion detection to multi-tenant container environments. In *IEEE European Dependable Computing Conference*, pages 133–136, Naples, Italy.

Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. (1996). A sense of self for Unix processes. In *IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, USA.

Hat, R. (2024). Docker: Desenvolvimento de aplicações em containers. `https://www.redhat.com/pt-br/topics/containers/what-is-docker`. Acessado em: 09/06/2024.

Heinrich, T., Will, N. C., Obelheiro, R. R., and Maziero, C. A. (2024). A categorical data approach for anomaly detection in WebAssembly applications. In *10th International Conference on Information Systems Security and Privacy*, pages 275–284, Rome, Italy.

Hodge, V. and Austin, J. (2004). A survey of outlier detection methodologies. *Springer Artificial Intelligence Review*, 22:85–126.

Huang, Y. and Zhang, Q. (2019). Identification of anomaly behavior of ships based on knn and lof combination algorithm. In *AIP Conference Proceedings*.

Jia, J., Zhu, Y., Williams, D., Arcangeli, A., Canella, C., Franke, H., Feldman-Fitzthum, T., Skarlatos, D., Gruss, D., and Xu, T. (2023). Programmable system call security with ebpf. *arXiv preprint arXiv:2302.10366*.

Jyothsna, V. and Rama Prasad, V. V. (2011). A review of anomaly based intrusion detection systems. *International Journal of Computer Applications*, 28(7):26–35.

Kubernetes (2024). Kubernetes components. `https://kubernetes.io/docs/concepts/overview/components`. Acessado em: 09/06/2024.

Levin, D. et al. (2024). strace: Linux syscall tracer. `https://strace.io`. Acessado em: 09/06/2024.

Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K., and Zhou, Q. (2018). A measurement study on Linux container security: Attacks and countermeasures. In *Annual Computer Security Applications Conference*, pages 418–429, San Juan, PR, USA.

LTTng (2024). Lttng: an open source tracing framework for linux. `https://lttng.org`.

Martínez-Magdaleno, S., Morales-Rocha, V., and Parra, R. (2021). A review of security risks and countermeasures in containers. *International Journal of Security and Networks*, 16(3):183–190.

Mishra, P., Varadharajan, V., Tupakula, U., and Pilli, E. S. (2018). A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE communications surveys & tutorials*, 21(1):686–728.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Platform, S. S. D. (2024). Sysdig: a universal system visibility tool with native support for containers:. `https://github.com/draios/sysdig`. Acessado em: 09/06/2024.

Rajagopalan, M., Hiltunen, M. A., Jim, T., and Schlichting, R. D. (2006). System call monitoring using authenticated system calls. *IEEE Transactions on Dependable and Secure Computing*, 3(3):216–229.

Randal, A. (2020). The ideal versus the real: Revisiting the history of virtual machines and containers. *ACM Computing Surveys*, 53(1):1–31.

Rocha, S. L., Nze, G. D. A., and de Mendonça, F. L. L. (2022). Intrusion detection in container orchestration clusters: A framework proposal based on real-time system call analysis with machine learning for anomaly detection. In *IEEE Iberian Conference on Information Systems and Technologies*, pages 1–4, Madrid, Spain.

Röhling, M. M., Grimmer, M., Kreubel, D., Hoffmann, J., and Franczyk, B. (2019). Standardized container virtualization approach for collecting host intrusion detection data. In *Federated Conference on Computer Science and Information Systems*, pages 459–463, Leipzig, Germany.

Rostedt, S. and Hat, R. (2014). Ftrace kernel hooks: more than just tracing. In *Linux Plumbers Conference*.

Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. *Springer Computer Science*, 2(3):160.

Shen, J., Zeng, F., Zhang, W., Tao, Y., and Tao, S. (2022). A clustered learning framework for host based intrusion detection in container environment. In *IEEE International Conference on Communications Workshops*, pages 409–414, Seoul, South Korea.

Srinivasan, S., Kumar, A., Mahajan, M., Sitaram, D., and Gupta, S. (2019). Probabilistic real-time intrusion detection system for Docker containers. In *Springer International Symposium on Security in Computing and Communications*, pages 336–347, Bangalore, India.

Sturm, R., Pollard, C., and Craig, J. (2017). *Application performance management (APM) in the digital enterprise: managing applications for cloud, mobile, iot and eBusiness*. Morgan Kaufmann.

Sultan, S., Ahmad, I., and Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7:52976–52996.

Tanenbaum, A. S. and Bos, H. J. (2016). *Sistemas Operacionais Modernos, 4ª Edição*. Pearson.

Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., and De Rose, C. A. (2013). Performance evaluation of container-based virtualization for high performance computing environments. In *IEEE Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240, Belfast, United Kingdom.

Zou, Z., Xie, Y., Huang, K., Xu, G., Feng, D., and Long, D. (2019). A Docker container anomaly monitoring system based on optimized isolation forest. *IEEE Transactions on Cloud Computing*, 10(1):134–145.