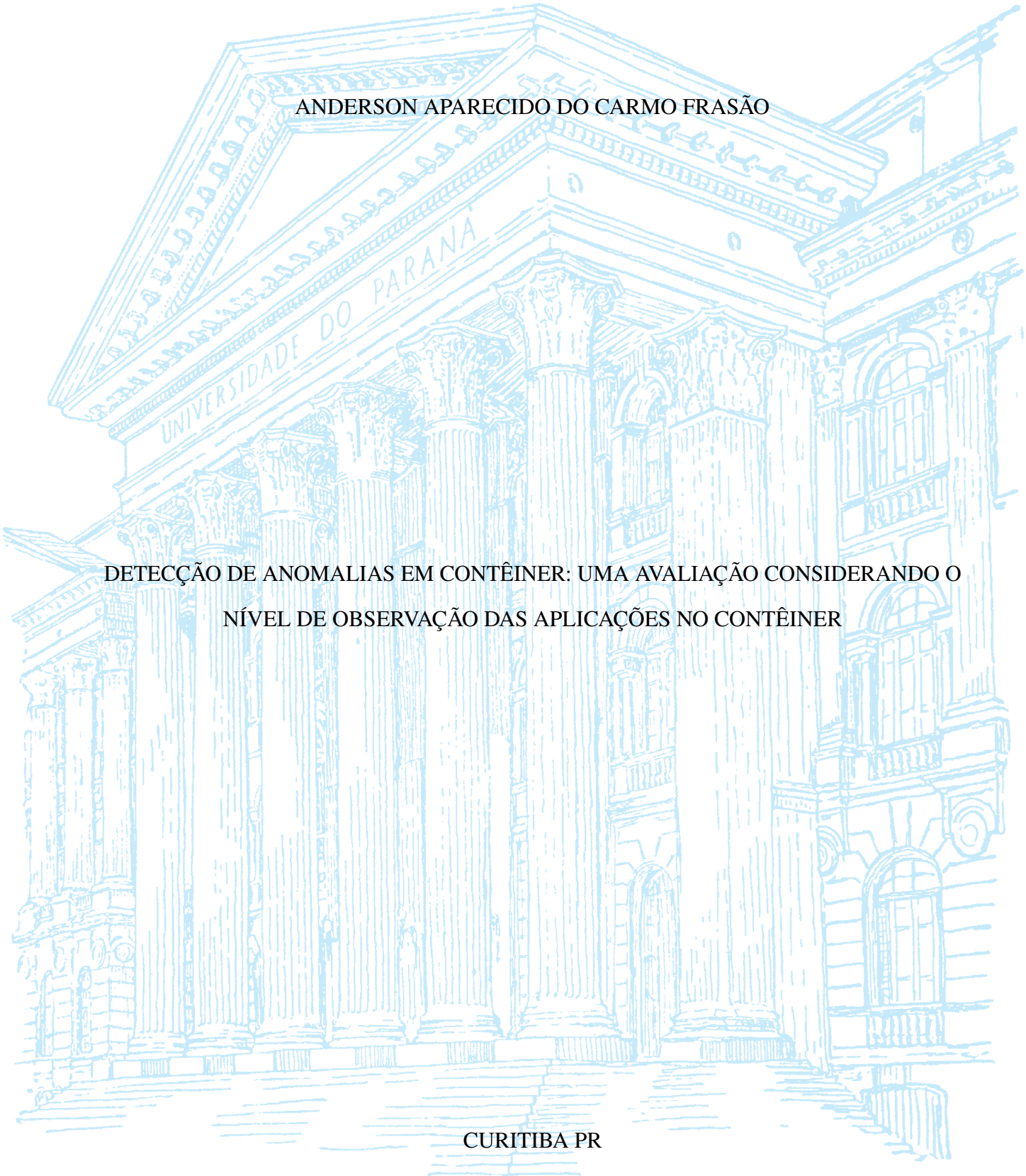
UNIVERSIDADE FEDERAL DO PARANÁ

ANDERSON APARECIDO DO CARMO FRASÃO



DETECÇÃO DE ANOMALIAS EM CONTÊINER: UMA AVALIAÇÃO CONSIDERANDO O NÍVEL DE OBSERVAÇÃO DAS APLICAÇÕES NO CONTÊINER

CURITIBA PR

2023

ANDERSON APARECIDO DO CARMO FRASÃO

DETECÇÃO DE ANOMALIAS EM CONTÊINER: UMA AVALIAÇÃO CONSIDERANDO O NÍVEL DE OBSERVAÇÃO DAS APLICAÇÕES NO CONTÊINER

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Carlos Alberto Maziero.

Coorientador: Tiago Heinrich.

CURITIBA PR

2023

Ficha catalográfica

Substituir o arquivo `0-iniciais/catalografica.pdf` pela ficha catalográfica fornecida pela Biblioteca da UFPR (PDF em formato A4).

Instruções para obter a ficha catalográfica e fazer o depósito legal da tese/dissertação (contribuição de André Hochuli, abril 2019):

1. Verificar se está usando a versão mais recente do modelo do PPGInf e atualizar, se for necessário (`https://gitlab.c3sl.ufpr.br/maziero/tese`).
2. conferir o Checklist de formato do Sistema de Bibliotecas da UFPR, em `https://portal.ufpr.br/teses_servicos.html`.
3. Enviar email para "`referencia.bct@gmail.com`" com o arquivo PDF da dissertação/tese, solicitando a respectiva ficha catalográfica.
4. Ao receber a ficha, inseri-la em seu documento (substituir o arquivo `0-iniciais/catalografica.pdf` do diretório do modelo).
5. Emitir a Certidão Negativa (CND) de débito junto a biblioteca (`https://www.portal.ufpr.br/cnd.html`).
6. Avisar a secretaria do PPGInf que você está pronto para o depósito. Eles irão mudar sua titulação no SIGA, o que irá liberar uma opção no SIGA pra você fazer o depósito legal.
7. Acesse o SIGA (`http://www.prppg.ufpr.br/siga`) e preencha com cuidado os dados solicitados para o depósito da tese.
8. Aguarde a confirmação da Biblioteca.
9. Após a aprovação do pedido, informe a secretaria do PPGInf que a dissertação/tese foi depositada pela biblioteca. Será então liberado no SIGA um link para a confirmação dos dados para a emissão do diploma.

Ficha de aprovação

Substituir o arquivo 0-iniciais/aprovacao.pdf pela ficha de aprovação fornecida pela secretaria do programa, em formato PDF A4.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais Gilmar Frasão e Marilene Honofre do Carmo, pela minha formação. Por me ensinarem, me apoiarem e por me incentivarem a trilhar o meu próprio caminho, estando ao meu lado sempre. À minha irmã Jaciara, por todo o companheirismo, todas as brigas e todas as cobranças, por acreditar sempre no melhor de mim, e por me ensinar muito mais do que eu posso listar.

Sou muito grato ao meu coorientador Tiago Heinrich, pela disponibilidade e pela enorme ajuda, paciência e orientação no desenvolvimento do trabalho, pelo apoio e incentivo em produzir um artigo derivados deste trabalho, que contribuíram muito na minha formação. Ao Prof. Dr. Carlos Alberto Maziero por ter me orientado ao longo de todo o percurso deste trabalho de graduação, e também por todo o conhecimento e ensinamentos compartilhados ao longo da minha graduação.

Sou grato ao Departamento de Informática (Dinf) da universidade, por toda a infraestrutura, recursos e oportunidades que tive o privilégio de usufruir ao longo desta jornada de graduação. À Universidade Federal do Paraná, por uma educação superior pública, gratuita e de qualidade.

Também sou muito grato a todos os amigos que me acompanharam ao longo desta jornada. Em especial ao Leonardo Camargo, por ter me ajudado na escolha desse curso, agradeço ao Gabriel Dolzan, meu primeiro amigo que a universidade me proporcionou, ao Dante Eleuterio, Eduardo Gobbo e Erick Eckermann, por estarem sempre ao meu lado, mesmo quando nem eu estava, ao Fernando Kiotheka por estar sempre disposto a me ajudar nos mais diversos problemas e duvidas que tive.

Finalmente, agradeço a todos que, em algum momento, tiveram que me escutar falando desse trabalho. E também a todos que de alguma forma, passaram pela minha vida nesse caminhada, por fim, meu muito obrigado.

RESUMO

A adoção crescente de tecnologias de virtualização ao longo da última década contribuiu com o desenvolvimento da virtualização baseada em contêineres. No entanto, o aumento do uso e implementação desses ambientes isolados também tem gerado preocupações quanto à sua segurança. Diante da proximidade entre o hospedeiro e o contêiner, surgem abordagens que utilizam detecção de intrusões através de anomalias como uma opção para monitorar e identificar comportamentos inesperados. Este estudo propõe o emprego de interações entre contêiner e Sistema Operacional para detectar ameaças em um ambiente de contêiner, sendo a extração de dados feita internamente ao ambiente, para diminuir a sobrecarga de dados e empregando algoritmos de Aprendizado de Máquina (ML) para distinguir entre comportamentos normais e anômalos. O Sisdig demonstrou ter potencial para detecção de anomalia, o f1 score (medida da precisão de um teste) indica em média 78.91% de precisão nos testes.

Palavras-chave: Detecção de Intrusão, Segurança de Computadores, e Contêineres.

ABSTRACT

The increasing adoption of virtualization technologies over the past decade has contributed to the development of container-based virtualization. However, the increased use and deployment of these isolated environments has also raised concerns about their security. In light of the proximity between the host and the container, approaches are emerging that use intrusion detection through anomalies as an option to monitor and identify unexpected behavior. This study proposes employing interactions between container and OS to detect threats in a container environment, with data extraction done internally to the environment to decrease data overload and employing Machine Learning (ML) algorithms to distinguish between normal and anomalous behaviors. Sisdig has shown potential for anomaly detection; the f1 score (a measure of the accuracy of a test) indicates on average 78.91% accuracy in the tests.

Keywords: Intrusion Detection, Computer Security, and Containers.

LISTA DE FIGURAS

2.1 Comparação entre VM e Contêiner, com base em (Docker, 2023). 16

4.1 Interação do *strace*, com base em (Sysdig, 2014). 24
4.2 Interação do *Sysdig*, com base em (Sysdig, 2014). 25
4.3 Estrutura para a coleta de dados com o *sysdig*. 27

5.1 Comparação entre a perspetiva de dados *strace* e *sysdig*.. 32
5.2 Diferença de Chamadas de Sistema Sysdig vs Strace (normal). 33
5.3 Diferença de Chamadas de Sistema Sysdig vs Strace (anormal) 34
5.4 As chamadas de sistema com 50 ou mais requisições usadas por interações mal-intencionadas. A comparação entre o *sysdig* e o *strace* demonstra a diferença entre o número de interações realizadas por aplicativos mal-intencionados 34

LISTA DE TABELAS

3.1 Trabalhos relacionados 23

5.1 Desempenho dos algoritmos de ML considerando todas as chamadas 31

LISTA DE ACRÔNIMOS

BoSC	Bag of System Calls
HIDS	Host-based Intrusion Detection System
IDS	Intrusion Detection System
ML	Machine Learning
MLP	Multilayer Perceptron
KNN	K-Nearest Neighbors
NIDS	Network Intrusion Detection System
RF	Random Forest
RCE	Remote Code Execution
SO	Sistema Operacional
STIDE	Sequence Time-Delay Embedding
SVM	Support Vector Machine
XSS	Cross-site Scripting

SUMÁRIO

1 INTRODUÇÃO 11
1.1 OBJETIVOS 12
1.2 METODOLOGIA 12
1.3 ORGANIZAÇÃO TEXTUAL 13
2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 SISTEMA DE DETECÇÃO DE INTRUSÃO 14
2.2 DETECÇÃO DE ANOMALIA 15
2.3 VIRTUALIZAÇÃO BASEADA EM CONTÊINER 16
2.3.1 Isolamento 18
2.3.2 Recursos do ambiente de container 18
2.4 MACHINE LEARNING 19
2.5 CONSIDERAÇÕES 20
3 REVISÃO BIBLIOGRÁFICA 21
3.1 DETECÇÃO DE INTRUSÃO EM CONTÊINERES 21
3.2 CONSIDERAÇÕES 23
4 PROPOSTA 24
4.1 PROBLEMÁTICA 24
4.2 ESTRATÉGIA 25
4.2.1 Coleta de Dados 26
4.3 MÉTODOS DE AVALIAÇÃO 26
4.4 CONSIDERAÇÕES 27
5 AVALIAÇÃO DE RESULTADOS 28
5.1 CONJUNTO DE TESTES 28
5.2 RESULTADOS 29
5.2.1 Estratégia utilizada 30
5.2.2 Detecção de Anomalias 30
5.2.3 Comparativo ponto de Observação 31
5.3 DISCUSSÃO 32
6 CONCLUSÃO 35
REFERÊNCIAS 36

1 INTRODUÇÃO

Os contêineres, pacotes de software que contêm todos os elementos necessários para serem executados em qualquer ambiente, oferecem vantagens em termos de desempenho e escalabilidade em comparação com maquinas virtuais. Sendo ideais para executar múltiplas aplicações com o mesmo sistema operacional, pois possuem menos sobrecarga, permitindo um maior número de instâncias a serem executadas em comparação com o recurso computacional utilizado por máquinas virtuais. (Joy, 2015) demonstra que os contêineres Linux mostraram-se superiores em termos de desempenho e escalabilidade, processando mais solicitações em menos tempo em comparação com as máquinas virtuais. No entanto, os contêineres têm um isolamento mais próximo ao sistema operacional do hospedeiro em relação às máquinas virtuais.

A introdução de aplicações em contêineres é um marco importante na evolução tecnológica. Os contêineres de virtualização são baseados no conceito de contêiner de transporte utilizado na indústria de frete, onde a carga é transportada de forma segura e eficiente em contêineres padronizados. O ambiente contêiner é uma tecnologia baseada em Linux que permite que várias instâncias de aplicativos compartilhem um único sistema operacional, eliminando a necessidade de cada instância ter seu próprio sistema operacional. Embora os contêineres não sejam uma tecnologia nova, eles ganharam destaque recentemente devido à demanda por implantação rápida de aplicativos. Os contêineres oferecem diversos benefícios, como portabilidade, menor consumo de memória, facilidade de migração e implantação rápida. Em comparação com a virtualização baseada em hipervisor, os contêineres compartilham o mesmo sistema operacional *host*, resultando na redução de recursos utilizados e permitindo que mais instâncias sejam hospedadas em um único servidor (Sturm et al., 2017).

Os riscos intrínsecos ao uso de contêineres estão relacionados a diversos fatores relacionados principalmente a implementação. Os **riscos da imagem** incluem componentes desatualizados, vulnerabilidades e inclusão de arquivos maliciosos. Os **riscos do orquestrador** envolvem práticas inadequadas de gerenciamento de contas e autenticação. Os **riscos do contêiner** incluem acesso ilimitado à rede e exploração de vulnerabilidades. Os **riscos do sistema operacional do host** são ampliados pelo uso de um *kernel* compartilhado. Os **riscos de implementação** estão relacionados à falta de segurança, configurações incorretas e vulnerabilidades em componentes específicos. Para a mitigação dos riscos associados com o uso de contêiner, são necessárias boas práticas de segurança, como atualizações regulares, controle de privilégios, monitoramento e adoção de políticas de segurança adequadas. Também é importante escolher imagens confiáveis e garantir a assinatura digital antes da implantação (Martínez-Magdaleno et al., 2021).

A ameaça atual no campo da tecnologia da informação requer o monitoramento e detecção precoce de ataques em sistemas (Röhling et al., 2019). Os Sistemas de Detecção de Intrusão Baseados em *Host* (HIDS) desempenham um papel crucial nesse processo, permitindo a inspeção das chamadas de sistema e a análise dos processos que acessam os sistemas. Os HIDS baseados em anomalias são particularmente eficazes na detecção de ataques desconhecidos, pois são treinados com comportamentos normais e identificam comportamentos anômalos durante um ataque. No entanto, a qualidade de um HIDS baseado em anomalias depende significativamente dos dados e da estratégia para a identificação de anomalias. Algumas das solução no meio de detecção de intrusão em contêiner são:

Integridade de instâncias do contêiner: O Amazon ECS oferece monitoramento de integridade para instâncias de contêiner, verificando problemas que possam impedir sua

execução. As verificações automatizadas ocorrem a cada dois minutos, retornando status de aprovação ou falha. Se todas as verificações forem aprovadas, o status geral será OK; caso contrário, será *IMPAIRED* (Amazon, 2023).

Análise de comportamento: Os contêineres Linux estabelecem comunicação com o núcleo do sistema operacional do hospedeiro por meio de chamadas de sistema. Ao observar as chamadas de sistema entre o contêiner e o núcleo do sistema operacional do hospedeiro, é viável compreender o comportamento do contêiner e identificar possíveis alterações que possam indicar uma tentativa de invasão contra o contêiner (Abed et al., 2015b).

Sistemas de prevenção de intrusão (IPS): São considerados como uma ampliação dos sistemas de Detecção de Intrusão (IDS - *Intrusion Detection System*), pois agem no sistema após a detecção de ameaças ou atividades maliciosas. (Lopez et al., 2014) propõe o BroFlow, um sistema de Detecção e Prevenção de Intrusão (IDPS) distribuído para a defesa contra ataques de negação de serviço (DoS).

Este trabalho de conclusão de curso apresenta um estudo de estratégias para a detecção de intrusão baseado em anomalia em um ambiente de virtualização em nível de contêiner. Considerando limitações de estudos anteriores como o ponto de observação dos contêiner em nível do sistema hospedeiro (Castanhel et al., 2021). A respectiva proposta foca em um processo de detecção que não esteja sendo prejudicado pelo ruído gerado pelo *container engine*, sem prejudicar o isolamento das aplicações em execução. O estudo consistiu em executar uma aplicação Wordpress em um contêiner Docker, com o objetivo de monitorar seu comportamento tanto em situações normais de uso quanto em casos de ataques ou operações maliciosas e coletar traços desse comportamento. Em seguida, foram testadas algumas técnicas de classificação para avaliar a eficiência da detecção de anomalia nesse contexto.

1.1 OBJETIVOS

Objetivo geral: Conduzir uma pesquisa focada na identificação de invasões por meio de anomalias em um ambiente de contêiner Docker.

Objetivos específicos:

- Avaliar a eficácia de um método de classificação de aprendizado de máquina na detecção de anomalias em contêiner;
- Investigar a viabilidade ao considerar o isolamento do contêiner e diferentes níveis de observação ao utilizar uma estratégia de detecção de anomalia dentro do contêiner;
- Criar uma base de dados que represente o comportamento da aplicação web em um ambiente de contêiner, com ênfase no comportamento anômalo; e
- Apresentar uma nova estratégia para a detecção de anomalias em um ambiente de contêiner.

1.2 METODOLOGIA

Primeiramente foi realizada uma investigação de ferramentas que permitiam a coleta de dados no ambiente de contêiner, sem prejudicar o isolamento da aplicação. Após, foi criado um conjunto de dados relacionados ao comportamento do contêiner, utilizando uma aplicação

web, mais especificamente o *Wordpress*. Para gerar esses dados foi feita uma coleção de testes reproduzindo tanto interações comuns, quanto interações maliciosas. Após sendo realizada a avaliação dos testes, foi feito o treinamento de um modelo de aprendizado de máquina, para então realizar testes sobre a identificação de anomalias em contêineres.

1.3 ORGANIZAÇÃO TEXTUAL

Neste trabalho, há uma divisão em seis partes. O capítulo 2 aborda a fundamentação teórica, explorando conceitos relacionados à detecção de anomalia, sistemas de detecção de intrusão, virtualização baseada em contêiner e diversos métodos de detecção. O capítulo 3 realiza uma revisão bibliográfica da literatura existente sobre detecção de anomalia. No capítulo 4 é apresentada a proposta, descrevendo a estratégia adotada, o processo de coleta e filtragem dos dados. O capítulo 5 apresenta a avaliação dos resultados obtidos e, por fim, o capítulo 6 traz a conclusão e considerações finais deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Na Seção 2.1 são apresentados os conceitos para detecção de intrusão. Na Seção 2.2 é apresentada detecção por anomalias. A Seção 2.3 apresenta sistemas de virtualização, destacando os ambientes de produção. A Seção 2.4 apresenta *Machine Learning* (ML) e seu uso para a detecção de ameaças. Na Seção 2.5, serão expostos as considerações dos trabalho feito.

2.1 SISTEMA DE DETECÇÃO DE INTRUSÃO

Existe uma crescente preocupação com a segurança na Internet e nos sistemas de computadores, devido ao aumento do uso de redes e das ameaças à segurança desses sistemas, (Liao et al., 2013). Invasão e vulnerabilidades em computadores ou sistemas de informação podem ter resultados desastrosos como violar políticas de segurança de computadores.

Sistemas de Detecção de Intrusão (IDS) são ferramentas usadas pelos administradores de segurança para detectar comportamentos ou ações intrusivas, efetuando a análise dos dados da auditoria e ajudando a estabelecer a culpabilidade do atacante, detectar a extensão dos danos e prevenir futuros ataques. O IDS é uma ferramenta valiosa tanto para análises em tempo real como após a ocorrência de um ataque (Laureano et al., 2003).

As três principais categorias de detecção de intrusão são: detecção baseada em assinatura (SD), detecção baseada em anomalia (AD) e análise de protocolo com estado (SPA). A detecção baseada em assinatura compara padrões de eventos capturados com padrões conhecidos de ameaças ou ataques, enquanto a detecção baseada em anomalia compara comportamentos observados com perfis normais previamente definidos (Liao et al., 2013).

Dentre os tipos de IDS existentes, há dois que se destacam, um deles é o Sistema de Detecção de Intrusão de Rede (NIDS) que utiliza mineração de dados para detectar intrusões em redes. Ele recebe dados capturados pelo dispositivo de rede, que coleta informações do cabeçalho de cada pacote. Esses dados são armazenados em arquivos e, em seguida, passam por uma filtragem para remover o tráfego indesejado (Vinchurkar e Reshamwala, 2012). Já o Sistema de Detecção de Intrusão de Hospedeiro (HIDS) é um sistema de detecção de intrusão que monitora as atividades em um *host*, como *logs* do sistema e do *shell*, para identificar comportamentos não autorizados. Ele utiliza métodos de mineração de dados, como redes neurais artificiais, para analisar os dados de auditoria do *host* e detectar ataques (Liu et al., 2018).

Uma técnica para a detecção de intrusão em *host* é baseado na observação de *logs* do sistema (Bridges et al., 2019). Esse tipo de análise é desafiador porque pode ser intensivo em termos de processamento e geralmente requer especialização humana. Alguns métodos destacados incluem o uso da teoria da probabilidade *naive-bayes*, técnicas de recuperação de informações e análise de *clustering* de usuários. Em geral, os resultados mostram que, embora a análise de *logs* possa produzir uma alta taxa de falsos positivos e muitas ameaças não detectadas, ela ainda é uma parte valiosa de um IDS.

Diversos estudos abordam a detecção de intrusão em sistemas de informação a partir de *logs* de auditoria. Os *logs* de auditoria são mais detalhados do que os *logs* de sistema e contêm informações como conexões de rede, ações em linha de comando, escalada de privilégios e alterações em arquivos. Pesquisadores utilizam diversas técnicas para a análise desses *logs*, incluindo modelos de Markov, lógica difusa e máquinas de vetores de suporte. Os resultados obtidos variam, mas mostram que a análise dos eventos de auditoria é uma técnica útil para

detectar atividades maliciosas. No entanto, a coleta e gerenciamento desses *logs* é cara e consome muitos recursos.

2.2 DETECÇÃO DE ANOMALIA

Detecção de anomalia 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). De acordo com (Lakhina et al., 2004), as anomalias são alterações significativas nos níveis de tráfego que afetam várias conexões.

A detecção de anomalias é uma tarefa importante de análise de dados que envolve a descoberta de padrões raros nos dados (Ahmed et al., 2016). As anormalidades são consideradas importantes porque indicam eventos significativos, mas raros, e podem levar a ações críticas em uma ampla gama de domínios de aplicação. Três classificações de detecção estatística de irregularidade são:

Modelo de mistura: Proposto por (Eskin, 2000), como uma abordagem para detectar anomalias em dados ruidosos. Nesse modelo, cada elemento é classificado como tendo uma pequena probabilidade λ de pertencer a uma classe anômala ou como pertencente à maioria dos elementos com probabilidade de 1-λ. Os autores assumem que os elementos com probabilidade de 1-λ representam o uso legítimo do sistema, enquanto as anormalidade têm probabilidade de λ.

Técnica de processamento de sinais: Método baseado em detecção de mudanças abruptas é apresentado por (Thottan e Ji, 2003), que descrevem anomalias como falhas de rede, problemas de desempenho e ataques de negação de serviço. O método utiliza bases de informações de gerenciamento para produzir uma função de saúde da rede e detectar comportamentos incomuns através de mudanças abruptas em suas estatísticas.

Análise de componentes principais (PCA): A técnica PCA usa uma combinação linear de variáveis aleatórias para criar componentes principais, que são descorrelacionados e ordenados em ordem decrescente de variância. Esses componentes podem ser usados para detecção de irregularidade com baixa complexidade computacional.

As principais técnicas para detecção de anomalias, incluindo métodos baseados em distância, densidade, modelo estatístico, aprendizado de máquina e mineração de dados (Hodge e Austin, 2004) são descritas a seguir:

Distância: Esta abordagem engloba a definição de uma medida de distância entre os pontos de dados e a identificação de pontos que estão distantes da maioria dos outros pontos. Tais como o *k-Nearest Neighbor* e o *Local Outlier Factor* (LOF).

Densidade: efetua a identificação de pontos de dados que têm uma densidade menor do que a dos pontos ao seu redor. Como o DBSCAN e o OPTICS, sendo limitados pela sensibilidade à escolha dos parâmetros de densidade e distância.

Estatístico: Esta abordagem engloba a modelagem da distribuição dos dados e a identificação de pontos que não se encaixam no modelo. Tais como o método de Boxplot e o método de Dixon.

Aprendizado de máquina: Esta abordagem engloba o uso de técnicas de aprendizado de máquina para identificar padrões nos dados. Algoritmos como o *Isolation Forest* e o *One-class SVM* aprendem uma classe de características e classificam padrões que diferem destes comportamentos como anomalia.

Mineração de dados: é responsável pela identificação de padrões anormais nos dados usando técnicas de mineração de dados, tais como associação de regras e agrupamento. Como o algoritmo de detecção de anomalias baseado em regras (ROD) e o algoritmo de agrupamento com detecção de anormalidade (COP).

2.3 VIRTUALIZAÇÃO BASEADA EM CONTÊINER

A virtualização baseada em contêineres aproveita os recursos do *kernel* para criar um ambiente isolado para os processos. Ao contrário da virtualização baseada em hipervisor, os contêineres não possuem seu próprio hardware virtualizado, mas utilizam o hardware do sistema hospedeiro. Isso significa que o software em execução nos contêineres se comunica diretamente com o kernel do hospedeiro e precisa ser executado no sistema operacional e na arquitetura da *Central Processing Unit* (CPU) em que o hospedeiro está sendo executado. A ausência da necessidade de emular hardware e inicializar um sistema operacional completo permite que os contêineres sejam iniciados rapidamente, em questão de milissegundos, e sejam mais eficientes do que as máquinas virtuais convencionais. As imagens dos contêineres geralmente são menores do que as imagens das máquinas virtuais, pois não precisam incluir uma cadeia completa de ferramentas para executar um sistema operacional, como *drivers* de dispositivo, *kernel* ou sistema de inicialização, (Eder, 2016).

Os hipervisores alocam hardware para máquinas virtuais (VMs). Os contêineres, por sua vez, virtualizam partes protegidas do sistema operacional, permitindo que vários contêineres compartilhem recursos sem perceber, devido à abstração de rede e processos individuais (Merkel et al., 2014).

Na figura 2.1 é ilustrada a comparação entre arquiteturas de virtualização baseada em contêiner e a virtualização baseada em hipervisor.

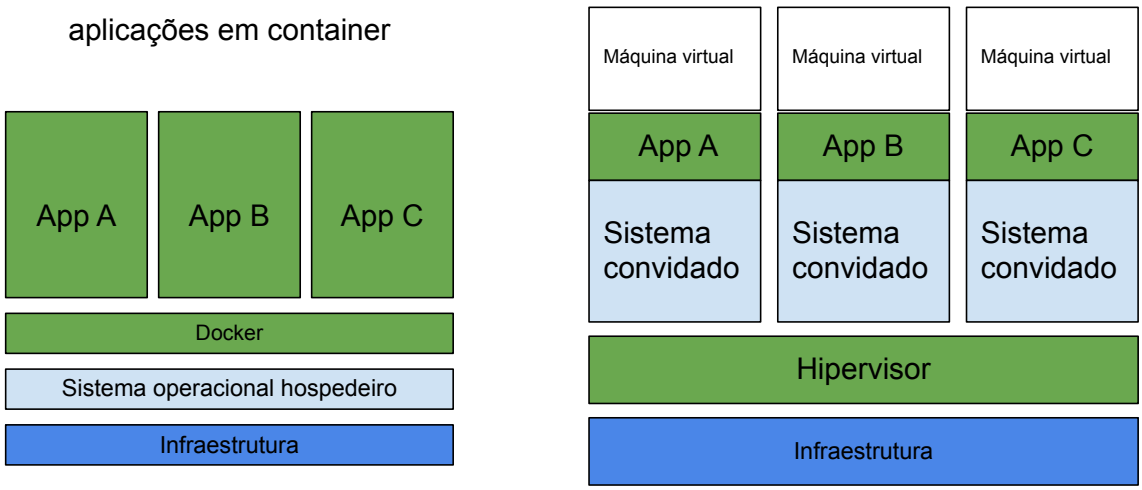


Figura 2.1: Comparação entre VM e Contêiner, com base em (Docker, 2023).

A virtualização baseada em contêineres cria instâncias isoladas do espaço do usuário em vez de máquinas virtuais completas. Essa forma de virtualização compartilha um único

kernel do sistema operacional entre as instâncias virtuais, o que resulta em um isolamento mais fraco em comparação com a virtualização baseada em hipervisor. Alguns exemplos de sistemas de virtualização baseados em contêineres (Xavier et al., 2013):

Linux-VServer: Implementações mais antigas de sistema baseado em contêiner Linux. Em vez de usar *namespaces* para garantir o isolamento, o Linux-VServer introduziu seus próprios recursos no kernel do Linux, como isolamento de processos, de rede e CPU, usando a chamada do sistema *chroot* para limitar o escopo do sistema de arquivos para processos. O isolamento de processo é realizado por meio de um espaço PID global. O Linux-VServer não virtualiza subsistemas de rede, mas usa identificadores de contêiner em pacotes para garantir que apenas o contêiner correto receba o tráfego. O Linux-VServer usa um esquema de *token bucket* para fazer isolamento de CPU. Os limites de recursos são estabelecidos por meio de chamadas de sistema e a ferramenta *rlimit*. O Linux-VServer é escalável, mas não suporta técnicas de virtualização usuais, como migração ao vivo, *checkpoint* e retomada. As versões recentes suportam *cgroups* e são gerenciadas pelo pacote de ferramentas *util-vserver*.

Docker: Uma ferramenta de linha de comando que permite criar, gerenciar e implantar contêineres. O Docker requer pouco conhecimento técnico em comparação com o uso do LXC, método de virtualização em nível de sistema operacional, para a mesma tarefa e oferece vários recursos essenciais para trabalhar com contêineres em um ambiente de produção. O Docker combina tecnologias para isolar processos com outras ferramentas úteis, facilitando a criação de imagens de contêiner baseadas em outras imagens de contêiner (Eder, 2016). De acordo com (RedHat, 2023a), o Docker é uma tecnologia que utiliza o *kernel* do Linux para criar e executar processos de forma independente em containers. Isso permite que vários processos e aplicativos sejam executados separadamente, maximizando o uso da infraestrutura e mantendo a segurança. O Docker usa um modelo de implantação baseado em imagem, facilitando o compartilhamento de aplicativos e serviços, incluindo todas as suas dependências. Além disso, o Docker automatiza a implantação de aplicativos em ambientes de containers. Essas ferramentas baseadas em containers Linux tornam o Docker fácil de usar, proporcionando aos usuários acesso a uma ampla variedade de aplicativos e controle total sobre versões e distribuição, além da capacidade de implantar rapidamente.

Kubernetes: É composto por diferentes componentes que trabalham juntos para criar e gerenciar um cluster. É uma plataforma de orquestração de contêineres de código aberto. Permite o gerenciamento automatizado de implantação, escalonamento e operação de aplicativos em contêineres. O Kubernetes fornece recursos avançados, como balanceamento de carga, gerenciamento de armazenamento, auto-recuperação e dimensionamento automático, tornando-se uma escolha popular para implantação e gerenciamento de aplicativos em grande escala. Funciona com contêineres Docker e outras tecnologias de contêineres, fornecendo uma camada de abstração adicional para simplificar a implantação e o gerenciamento de aplicativos distribuídos (Kubernetes, 2023).

A tecnologia de virtualização de nível de sistema operacional conhecida como Linux contêiner permite isolar e conter um ou mais processos em um ambiente que aparenta ser o sistema inteiro, mas que na verdade compartilha o *kernel* Linux com outros containers. A tecnologia utiliza dois mecanismos principais, *namespace* e *cgroup*, para fornecer isolamento e limitar o uso de recursos, como CPU e memória. No entanto, a interdependência e a relação

de influência mútua entre esses mecanismos de segurança do *kernel* podem fazer com que prejudiquem sua capacidade de proteção (Lin et al., 2018).

2.3.1 Isolamento

Os *namespaces* do *kernel* Linux são usados para separar processos uns dos outros, criando diferentes espaços de nomes para processos que precisam ser isolados. Os *namespaces* permitem a criação de contêineres contendo processos com identificadores de processo (PID) que já estão sendo usados no sistema hospedeiro ou em outros contêineres. Isso facilita a migração de contêineres suspensos.

Os grupos de controle (*cgroups*) são usados para rastrear processos e grupos de processos, permitindo o controle e a limitação flexível de recursos. Embora não sejam obrigatórios para isolar processos, os *cgroups* fornecem um mecanismo para controlar e gerenciar o uso de recursos, garantindo que recursos físicos não sejam desperdiçados ou indisponíveis devido a outros processos.

O *Mandatory Access Control* (MAC) é um conjunto de conceitos que define a gestão do controle de acesso e é usado para fortalecer os mecanismos de controle de acesso no Linux/Unix, que permite restringir o acesso a recursos sensíveis que não precisam ser acessados em determinado contexto.

Os *namespaces* do *kernel* e os *cgroups* são fundamentais para a virtualização baseada em contêineres pois fornecem isolamento de processos e recursos, portabilidade, eficiência e escalabilidade. A aplicação dessas ferramentas de segurança aos contêineres adiciona mais uma camada de proteção para mitigar ataques contra o hospedeiro e outros contêineres de dentro do contêiner.

2.3.2 Recursos do ambiente de container

Desde a criação de imagens personalizadas até a orquestração eficiente de containers em *clusters*, os recursos disponíveis oferecem um amplo espectro de possibilidades para otimizar a produtividade e a confiabilidade no ciclo de vida de uma aplicação. Os ambientes de contêiner fornecem diversos recursos, como o isolamento de recursos, a portabilidade entre ambientes, a gestão de dependências, a escalabilidade horizontal, a integração contínua e a implantação contínua.

O Docker é uma plataforma de código aberto criada pela Docker.Inc, usando a linguagem Go. Com alta performance, essa ferramenta facilita a criação e gestão de ambientes isolados, permitindo a rápida disponibilização de programas aos usuários finais. A seguir é apresentado algumas características de Docker:

Modularidade: A abordagem do Docker para a conteinerização permite desativar partes de uma aplicação para reparo ou atualização sem interromper completamente sua execução. Além disso, o Docker possibilita o compartilhamento de processos entre várias aplicações, seguindo uma abordagem semelhante à arquitetura orientada a serviços (SOA).

Camadas e controle de versão de imagens: O Docker utiliza camadas para compor suas imagens, permitindo a reutilização e aceleração do processo de criação de containers. Cada modificação na imagem gera uma nova camada, otimizando velocidade, tamanho e eficiência. Estas mudanças são registradas, proporcionando controle completo sobre as imagens do contêiner.

Reversão: É possível recuperar a versão anterior usando a função de reversão. Isso acontece devido às camadas criadas no ambiente contêiner. Esse processo é ainda mais eficiente porque é compatível com a abordagem de desenvolvimento ágil. Assim, a equipe pode contar facilmente com as práticas de integração e implantação contínua, mantendo a eficiência no desenvolvimento da aplicação.

Implantação rápida: A implantação de *software* costumava ser um processo demorado que levava dias. No entanto, com os containers Docker, é possível implantar em apenas alguns segundos. Ao criar um contêiner para cada aplicação, é possível compartilhá-los facilmente com novas aplicações. Não é necessário reiniciar o sistema operacional para adicionar ou mover um contêiner, o que reduz significativamente o tempo de implantação. Além disso, é fácil e econômico criar e destruir dados associados aos containers, sem qualquer preocupação.

Além do Docker, o Kubernetes oferece várias vantagens ao ambiente de desenvolvimento e implementação de aplicativos na nuvem. A principal vantagem é a capacidade de programar e executar containers em clusters de máquinas virtuais ou físicas de forma eficiente. Isso permite a otimização do desenvolvimento de apps para a nuvem e o uso de uma infraestrutura baseada em containers confiável para ambientes de produção (RedHat, 2023b).

Além disso, o Kubernetes possui outros recursos desejáveis. Tais como: (i) A possibilidade de orquestrar containers em vários *hosts*; (ii) Maximizar o uso de *hardware* para executar *apps* empresariais; (iii) Controlar e automatizar atualizações e implantações de aplicações; (iv) Adicionar armazenamento para *apps stateful*; (v) Escalar aplicações e recursos correspondentes rapidamente; (vi) Gerenciar serviços de forma assertiva para garantir a execução adequada das aplicações; (vii) Permitir a autorrecuperação e verificação de integridade das *apps* por meio de automação. Desse modo, este software é especialmente útil para implementação de vários contêineres e/ou vários *hosts*.

2.4 MACHINE LEARNING

A aprendizagem é um fenômeno complexo que envolve a assimilação de conhecimento, o aprimoramento de habilidades e a descoberta de novas informações. Desde a introdução dos computadores, os pesquisadores têm buscado capacitar as máquinas com essas habilidades, representando um desafio cativante no campo da inteligência artificial. A aprendizagem de máquina é a área de estudo dedicada a investigar e modelar esses processos de aprendizagem.

Os objetivos da aprendizagem de máquina incluem a melhoria do desempenho em tarefas específicas, a simulação do aprendizado humano e a exploração de métodos e algoritmos de aprendizagem. Embora cada objetivo tenha sua ênfase particular, o avanço em um geralmente impulsiona o progresso nos outros. Um exemplo disso é quando a pesquisa sobre a aprendizagem humana orienta a exploração de diferentes métodos de aprendizagem. Além disso, a análise teórica pode oferecer perspectivas valiosas para pesquisas no campo da psicologia. Essa interação desafia e fortalece o campo da inteligência artificial como um todo (Carbonell et al., 1983).

Existem várias categorias nas quais os algoritmos de aprendizado de máquina são agrupados, algumas delas são apresentadas a seguir (Nasteski, 2017):

Aprendizado supervisionado: Os diferentes algoritmos produzem uma função que relaciona as entradas aos resultados desejados. Uma forma comum de descrever a tarefa de aprendizado supervisionado é o problema de classificação, no qual o objetivo é que o algoritmo aprenda a função que associa um vetor a uma das diversas classes. Isso é

realizado através da observação de vários exemplos de entrada e saída dessa função, buscando se aproximar do seu comportamento.

Aprendizado não supervisionado: Cria um modelo para um conjunto de entradas onde exemplos rotulados não estão disponíveis.

Aprendizado semi supervisionado: Utiliza tanto exemplos rotulados quanto não rotulados em conjunto para gerar uma função ou classificador adequado.

Aprendizado por reforço: O algoritmo adquire uma política de ação com base na observação do ambiente. Cada ação realizada pelo algoritmo causa um impacto no ambiente, e o ambiente fornece um *feedback* que orienta o processo de aprendizado do algoritmo.

Com a grande quantidade de dados produzidos diariamente, é necessário atualizar constantemente os modelos de ML para lidar com ameaças emergentes. O uso de métodos totalmente supervisionados é problemático devido ao desequilíbrio extremo entre as classes de ataques e exemplos normais, o que torna a rotulação dos dados um gargalo no processo de aprendizado. Além disso, é mostrado um esquema para desenvolver e avaliar soluções de aprendizagem automática para a cibersegurança, esse esquema inicia com coleta de dados, em seguida é feita a extração de atributos, extração de características, após essa etapa, há a opção de treinar um modelo ou ir para os testes. Os testes tem duas etapas, a avaliação e a atualização do modelo (Ceschin et al., 2020).

O Aprendizado de Máquina é um campo que usa experiência e dados para melhorar o desempenho e fazer previsões precisas. Os algoritmos de aprendizagem são projetados para serem eficientes e precisos, levando em consideração a complexidade dos dados e a quantidade de amostras necessárias (Mohri et al., 2018). A aprendizagem de máquina combina conceitos de ciência da computação, estatística, probabilidade e otimização para análise de dados e tomada de decisões.

2.5 CONSIDERAÇÕES

Com as seções 2.1 a 2.4, foi gerado um sistema de detecção de intrusão, além disso, foi feita a comparação entre dois pontos de vista para a coleta de dados de um sistema de virtualização basada em contêiner.

3 REVISÃO BIBLIOGRÁFICA

A Seção 3.1 apresenta a revisão da literatura voltada à detecção de intrusão no ambiente de contêiner, bem como suas limitações e desafios. A Seção 3.2 faz uma revisão geral sobre os principais pontos vistos em cada artigos presente na revisão bibliográfica.

3.1 DETECÇÃO DE INTRUSÃO EM CONTÊINERES

No trabalho (Abed et al., 2015a) é apresentado um sistema de detecção de intrusão baseado em contêineres para aplicativos em execução no Linux. O IDS usa um modelo de perfil de comportamento do aplicativo para detectar atividades maliciosas em tempo real. O sistema coleta informações sobre as chamadas de sistema feitas pelo aplicativo em execução no contêiner e usa essas informações para criar um modelo de comportamento normal. Quando o aplicativo faz uma chamada de sistema que não está dentro do modelo normal, o IDS identifica essa atividade como suspeita e gera um alerta.

(Du et al., 2018) propõem uma abordagem para detectar e diagnosticar anomalias em microsserviços baseados em contêineres, usando monitoramento de desempenho. A abordagem usa uma combinação de análise de séries temporais e aprendizado de máquina para detectar anomalias em microsserviços. As métricas de desempenho dos microsserviços, como uso de CPU e memória, são coletadas e usadas para criar um modelo de comportamento normal do microsserviço. Quando o microsserviço se desvia do modelo, a atividade é considerada suspeita e pode ser uma anomalia. Também é apresentado um método para diagnosticar anomalias usando árvores de decisão. O sistema identifica quais métricas são mais relevantes para a anomalia em questão e fornece uma explicação para a causa da anomalia.

O trabalho de (Sultan et al., 2019) aborda os desafios e problemas de segurança relacionados ao uso de contêineres na computação em nuvem. Entre os principais desafios de segurança enfrentados pelas organizações que utilizam contêineres, estão a segurança do *host*, gerenciamento de identidade e acesso, segurança do registro de contêineres e segurança dos próprios contêineres. Algumas das soluções existentes para proteger contêineres são *namespaces*, *cgroups*, uso de plataformas confiáveis, entre outros. Outro aspecto importante para a proteção de contêineres é a colaboração entre as equipes de segurança e desenvolvimento para garantir a segurança ao longo do ciclo de vida dos contêineres.

Em (Zou et al., 2019) é apresentado um sistema de monitoramento de anomalias para contêineres Docker baseado na técnica de *Isolation Forest* otimizada. O objetivo é detectar anomalias em contêineres Docker em tempo real, garantindo a segurança e o desempenho do sistema. A técnica de *Isolation Forest* (iForest) é um método eficaz para detecção de anomalias em dados não supervisionados, podendo ser adaptada para o ambiente de contêineres Docker considerando as peculiaridades deste ambiente e pode ser usada para detectar anomalias em tempo real. Resultados de testes mostram que o sistema é capaz de detectar anomalias com alta precisão e eficiência em ambientes de contêineres Docker. Além disso, o sistema proposto apresenta desempenho superior em termos de precisão e eficiência, comparado com uma implementação baseada em iForest e outra baseada em densidade.

(Flora e Antunes, 2019) analisa a eficácia de diferentes abordagens de detecção de intrusão em ambientes de contêineres *multi-tenant*. O estudo avaliou três abordagens de detecção de intrusão: detecção de anomalias baseada em rede, detecção de anomalias baseada em comportamento e detecção de assinaturas. Os resultados mostraram que a detecção de anomalias

baseada em comportamento foi a abordagem mais eficaz para detectar ataques em contêineres *multi-tenant*. O artigo também discute possibilidades para futuras pesquisas, como o uso de técnicas de aprendizado de máquina para aprimorar a detecção de anomalias em contêineres *multi-tenant*.

O artigo (Srinivasan et al., 2019) apresenta uma solução para a detecção de intrusões em tempo real em contêineres Docker. A arquitetura da solução proposta utiliza dados de monitoramento de contêineres para construir um modelo probabilístico baseado em redes Bayesianas. Esse modelo é treinado usando n-gramas de chamadas do sistema e a probabilidade de ocorrência desses n-gramas é calculada e, posteriormente, é capaz de avaliar a probabilidade de comportamentos anormais em tempo real.

O estudo realizado por (Castanhel et al., 2020b) aborda a detecção de anomalias em ambientes de contêineres, explorando o impacto da visão parcial das informações nos resultados. O objetivo é investigar como o tamanho da janela afeta a detecção de anomalias nesse contexto. Os dados consistem em chamadas e sinais emitidos pelos contêineres, representando comportamentos normais e anômalos que exploram vulnerabilidades de *Remote Code Execution* (RCE). Dois métodos de Classificação do tipo Classe Única (OCC) foram avaliados: *Isolation Forest* e *One-Class SVM*. Foram realizados experimentos com dois tipos de conjuntos de dados: um com dados brutos e outro com dados filtrados, removendo chamadas de sistema classificadas como de baixo nível de ameaça. A avaliação considerou o crescimento do tamanho da janela de treinamento e teste. Os resultados mostraram que tanto os dados brutos quanto os dados filtrados alcançaram resultados razoáveis com janelas pequenas, e houve variação nos resultados à medida que o tamanho da janela aumentou. O algoritmo *Isolation Forest* obteve melhores resultados conforme o tamanho da janela aumentou, superando o desempenho da *One-Class SVM*.

No trabalho subsequente, (Castanhel et al., 2021) houve foco em uma aplicação executada em um ambiente de contêiner e observada a partir do hospedeiro, com o objetivo de explorar técnicas de machine learning para identificar ataques e entender como os sistemas de detecção de intrusão se comportam com essa perspectiva externa. Foram apresentadas três abordagens para coletar informações das chamadas de sistema dos processos dentro de um contêiner: (i) executar o Sistema de Detecção de Intrusão (IDS) no hospedeiro; (ii) usar uma aplicação de rastreio dentro do mesmo contêiner da aplicação; e (iii) utilizar um contêiner privilegiado para a execução do IDS. A primeira abordagem foi utilizada coletando os dados das chamadas de sistema com a ferramenta *strace*. Foram realizados testes com um protótipo da proposta, utilizando o *Wordpress* como aplicação de teste devido à sua popularidade e ao fato de possuir diversas vulnerabilidades conhecidas. Foram testados diferentes tamanhos de janela e quatro algoritmos de classificação: *K-Nearest Neighbors* (KNN), *Random Forest* (RF), *Multilayer Perceptron* (MLP) e *AdaBoost* (AB). Os resultados foram avaliados utilizando métricas como precisão, *recall* e *f1-score*. Os resultados mostraram que a proposta é eficaz na detecção de anomalias.

O artigo (Rocha et al., 2022) propõe um *framework* que permite a implementação de HIDS baseado em anomalias de chamada de sistema, especificamente em ambientes que fazem uso de plataformas para execução de contêineres. Os contêineres vêm tomando espaço no mercado e assim é necessário se preocupar com a segurança desses ambientes. A arquitetura implementada permite o acompanhamento e a investigação eficaz de eventos de segurança em contêineres, com ênfase na detecção de intrusões e na identificação de anomalias. Essa implementação se mostrou viável e funcional para implementar em produção.

(Shen et al., 2022) aborda a tecnologia de contêineres e sua vulnerabilidade a ataques de segurança, propondo uma nova estrutura de detecção de anomalias utilizando algoritmo de cluster. O artigo destaca que a detecção de intrusão baseada em anomalias é uma abordagem eficaz para a detecção de comportamentos anômalos de contêineres, e que o novo método

proposto pode identificar recipientes construídos na mesma imagem de aplicação sem rotulagem manual, reduzindo a Taxa de Falso Positivo da detecção de anomalias e aumentando a Taxa de Positivo Verdadeiro em comparação com o método tradicional. A análise estática de imagens de contêineres e a detecção de anomalias de um contêiner em funcionamento também é discutida, sendo as principais abordagens utilizadas para a segurança de contêineres atualmente.

Trabalho	**Estratégia utilizada na coleta de dados**	**Ambiente**	***Dataset***
(Abed et al., 2015b)	Strace	Contêiner Docker	-
(Du et al., 2018)	cAdvisor, Heapster, InfluxDB e Grafana	Contêiner Kubernetes	Dados próprios
(Zou et al., 2019)	Ganglia, Nagios, Akshay, cAdviosr e InfluxDB	Contêiner Docker	-
(Flora e Antunes, 2019)	Strace e sysdig	Contêiner (Docker e LXC)	-
(Sultan et al., 2019)	Starlight, OpenWhisk	Contêiner Docker	-
(Srinivasan et al., 2019)	N-grams para análise probabilística	Contêiner Docker	Dataset UNM
(Castanhel et al., 2021)	Strace	Contêiner Docker	Dados próprios
(Rocha et al., 2022)	Sysdig	Contêiner Kubernetes	Dados próprios
(Shen et al., 2022)	Sysdig	Contêiner (Docker e LXC)	-

Tabela 3.1: Trabalhos relacionados

A Tabela 3.1 mostra, em resumo, na coluna Estratégia utilizada, os métodos implementados para coletar os dados necessários para os testes, na coluna Ambiente, o modelo de contêiner de onde esses dados foram coletados e na coluna *Dataset*, qual o conjunto de dados utilizado para os testes, para cadas trabalho revisado na subseção anterior.

3.2 CONSIDERAÇÕES

A revisão bibliográfica apresentada aborda várias técnicas para a detecção de anomalias e intrusões em ambientes de contêineres Linux, destacando a importância da segurança nesse contexto. Foram apresentados sistemas de detecção baseados em chamadas de sistema, perfil de comportamento do aplicativo e monitoramento de desempenho, além de discussões sobre os desafios de segurança enfrentados pelas organizações que utilizam contêineres na computação em nuvem.

Os artigos apresentados também abordam a eficácia de diferentes abordagens de detecção de intrusão, como detecção de anomalias baseada em rede, comportamento e assinaturas, e destacam a importância do uso de técnicas de aprendizado de máquina para aprimorar a detecção de anomalias em contêineres multi-inquilinos. Algumas limitações foram identificadas nos trabalhos avaliados, dentre elas, podemos incluir questões como eficiência ou precisão insatisfatórias, falta de integração com ferramentas específicas, complexidade nas soluções propostas, foco limitado em vulnerabilidades específicas, e possíveis desafios na implementação prática ou escalabilidade.

4 PROPOSTA

O capítulo 4 apresenta a proposta da pesquisa. A Seção 4.1 abrange as problemáticas na área. A Seção 4.2 aborda a estratégia utilizada. A seção 4.3 apresenta o processo de avaliação realizado. Por fim, na seção 4.4 é feita algumas considerações sobre o capitulo.

4.1 PROBLEMÁTICA

O problema central abordado neste trabalho é a falta de uma solução abrangente e eficaz para a detecção de anomalias e intrusões em contêineres Docker. As soluções tradicionais de HIDS se concentram em monitorar as atividades do sistema operacional hospedeiro, sem levar em consideração o ambiente contido no contêiner em si. Isso limita a capacidade de detecção de ameaças específicas aos contêineres e torna a resposta a incidentes menos precisa.

Além disso, a maioria das soluções com base em HIDS estão limitadas ao uso de chamadas de sistema para detecção de intrusões. Estas chamadas são coletadas sem considerar o impacto que o ambiente de contêineres Docker pode estar tendo no contêiner em execução. As características únicas dos contêineres, como o uso de *namespaces* para o isolamento de recursos, requerem uma abordagem específica para a detecção de anomalias que considere o ambiente de isolamento.

Uma ferramenta conhecida para coleta de informações por chamadas de sistema é o *strace*, que utiliza a chamada de sistema *ptrace*, fornecida pelo Linux e outros sistemas operacionais, para instrumentar um processo-alvo e "escutar" as chamadas de sistema desse processo. No momento que 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, como mostrado na figura 4.1. No entanto, essa abordagem possui uma desvantagem significativa: cada mudança de contexto original é transformada em várias mudanças de contexto, o que resulta em um desempenho ineficiente e um processo rastreado aguardando o *strace* para realizar sua decodificação, (Sysdig, 2014).

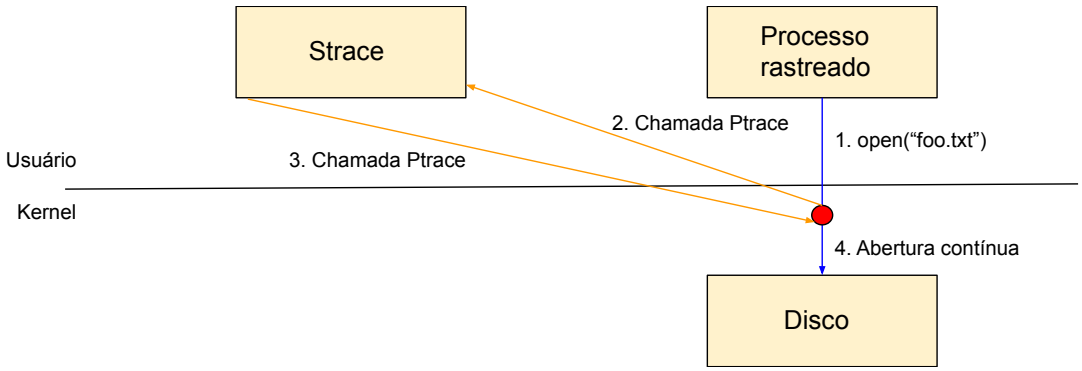


Figura 4.1: Interação do *strace*, com base em (Sysdig, 2014).

Por outro lado, o *sysdig* 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, como mostrado na figura 4.2. 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, (Sysdig, 2014).

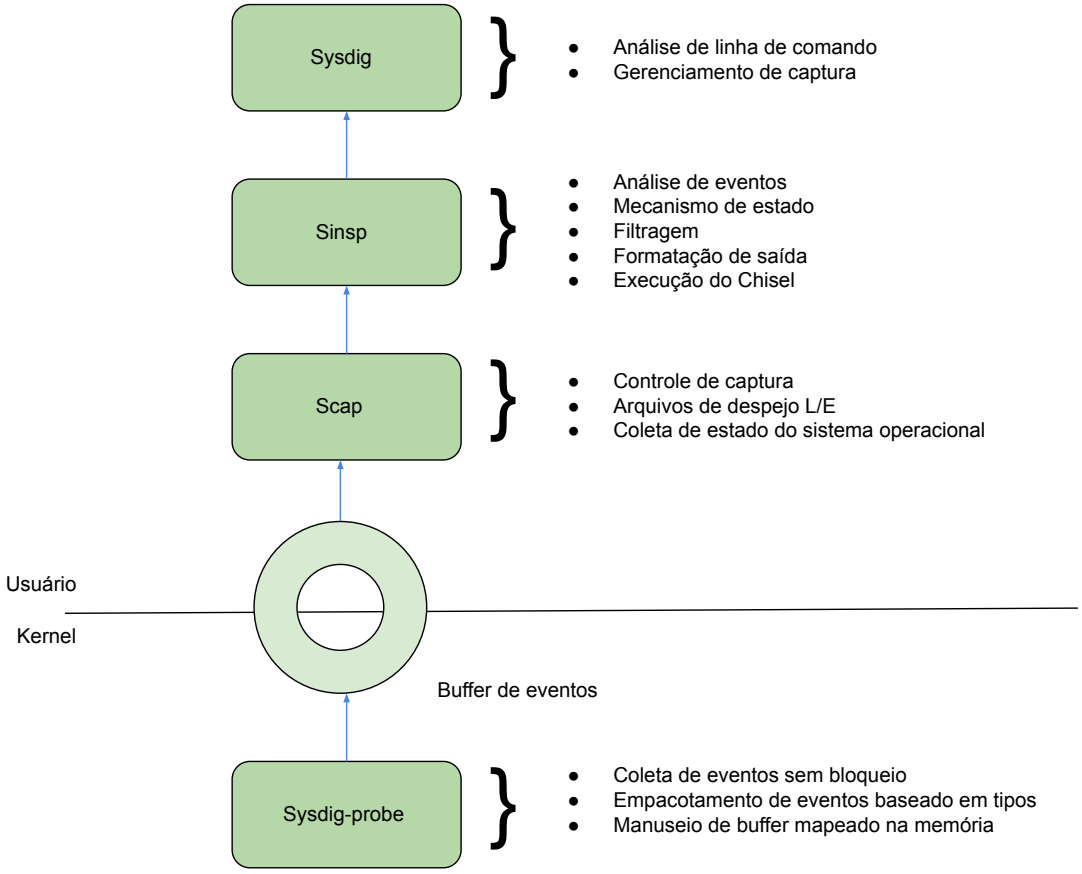


Figura 4.2: Interação do *Sysdig*, com base em (Sysdig, 2014).

O *Strace* é conhecido por ter um *overhead* significativo, pois sua abordagem envolve a interceptação de todas as chamadas de sistema de um processo e a exibição dos detalhes para fins de depuração. Isso pode resultar em uma degradação perceptível do desempenho do sistema, especialmente quando usado em processos intensivos em I/O (entrada/saída) (Sysdig, 2014). Por outro lado, o *Sysdig* utiliza uma abordagem diferente para monitorar chamadas de sistema. Ele faz uso de uma funcionalidade do *kernel* Linux chamada *extended Berkeley Packet Filter* (eBPF) (Sysdig, 2019) para coletar dados em tempo real sobre as atividades do sistema. Essa abordagem é mais eficiente e geralmente tem um impacto menor no desempenho.

É importante notar que os conjuntos específicos de chamadas variam entre as duas perspectivas de observação. Essa variação pode complicar o processo de identificação de anomalias, especialmente se o modelo utilizar diferentes pontos de observação simultaneamente. A detecção de anomalias baseia-se na identificação de operações que se desviam do comportamento normal, tornando a consistência na observação crucial para uma análise precisa.

4.2 ESTRATÉGIA

A proposta deste trabalho é desenvolver um HIDS especializado para a detecção de anomalias e intrusões em contêineres Docker. Através do uso de chamadas de sistemas e características internas do contêiner, o isolamento da aplicação foi levado em consideração e

a coleta das informações foi realizada em um nível próximo da aplicação para que não seja captura informações que não pertencem ao contêiner. A abordagem levou em consideração as características específicas dos contêineres Docker, como o isolamento de recursos e o uso de *namespaces*, para garantir uma detecção precisa e eficiente.

A solução proposta tem as seguintes etapas principais:

1. Coleta de dados: foi realizada a coleta das chamadas de sistema realizadas dentro do contêiner Docker, capturando informações relevantes sobre as atividades em execução.

2. Pré-processamento: Os dados coletados serão pré-processados para extrair características relevantes e reduzir o ruído, garantindo que apenas informações significativas sejam consideradas no processo de detecção.

3. Treinamento: Utilizando técnicas de aprendizado de máquina, o HIDS foi treinado em um conjunto de dados anotados, onde foram identificados os padrões normais de comportamento do contêiner.

4. Detecção de anomalias: Com base no modelo treinado, o HIDS foi capaz de detectar anomalias e intrusões no comportamento do contêiner em tempo real. Qualquer desvio significativo em relação aos padrões estabelecidos foi considerado uma possível ameaça.

4.2.1 Coleta de Dados

Nesta subseção, é descrita a abordagem utilizada para coletar os dados relevantes. Para essa finalidade, foi empregada a ferramenta *SysDig*, que é uma ferramenta de monitoramento de contêineres e sistemas operacionais que fornece uma visão detalhada do desempenho e do comportamento do sistema, (Sysdig, 2017).

Através do *SysDig*, pode-se capturar métricas essenciais do sistema, como utilização de CPU, memória e E/S de disco, além de informações relacionadas a eventos e chamadas do kernel.

Cada linha do arquivo de rastreamento produzido pelo *sysdig* contém o Identificador de evento, produzido pela ferramenta, o *Timestamp* da chamada, a *Thread* que está processando a solicitação, nome e ID do contêiner, nome e ID do processo do aplicativo que está se comunicando com o contêiner/kernel, o nome da *System call* e seus parâmetros; esse formato é mostrado na Listagm 4.1.

```
1 1448963 15:18:28.376406479 1 wordpress (cdab6c786756) apache2 (337432:74) > set_robust_list
2 1448964 15:18:28.376406952 1 wordpress (cdab6c786756) apache2 (337432:74) < set_robust_list
3 1448973 15:18:28.376449921 1 wordpress (cdab6c786756) apache2 (337432:74) > getpid
4 1448974 15:18:28.376450361 1 wordpress (cdab6c786756) apache2 (337432:74) < getpid
5 1448975 15:18:28.376452360 1 wordpress (cdab6c786756) apache2 (337432:74) > mmap addr=0 length=134217728 prot=0(PROT_NONE)
      flags=74(MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE) fd=-1(EPERM) offset=0
6 1448976 15:18:28.376456877 1 wordpress (cdab6c786756) apache2 (337432:74) < mmap res=7F7959DFF000 vm_size=535852 vm_rss
      =56496 vm_swap=0
7 1448977 15:18:28.376458138 1 wordpress (cdab6c786756) apache2 (337432:74) > munmap addr=7F7959DFF000 length=35655680
8 1448978 15:18:28.376463936 1 wordpress (cdab6c786756) apache2 (337432:74) < munmap res=0 vm_size=501032 vm_rss=56496
      vm_swap=0
```

Listing 4.1: Logs retirados a partir do *sysdig*.

Para utilizar os dados, foram mantidas somente o nome das *system calls*, pois para os testes só utilizamos as quantidades de vezes que cada chamada foi feita.

4.3 MÉTODOS DE AVALIAÇÃO

Com o intuito de avaliar uma técnica 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 nessa perspectiva, o sistema gerado esta representado na Figura 4.3.

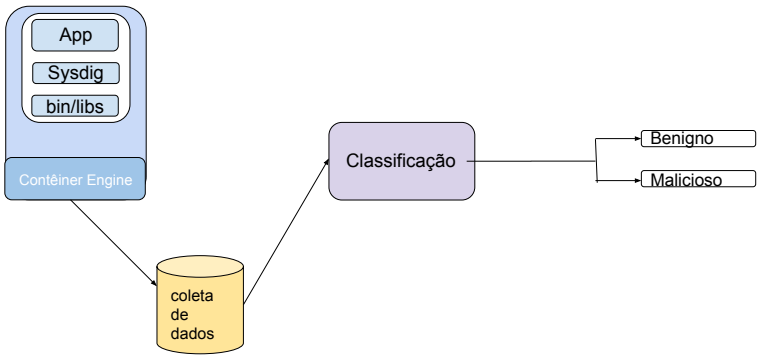


Figura 4.3: Estrutura para a coleta de dados com o *sysdig*.

4.4 CONSIDERAÇÕES

A proposta de detecção de anomalias apresentada neste capítulo foca no uso de dados para representar o comportamento de aplicações em contêineres. Conforme apresentado na Figura 4.3, a estrutura faz a coleta dos dados, presentes na Listagem 4.1, e a classificação dos mesmos utilizando algoritmos de *Machine Learning*. Após uma análise aprofundada da representação, espera-se obter uma compreensão mais clara da forma como os dados são empregados para garantir a segurança e identificar anomalias em aplicações executadas em contêineres.

5 AVALIAÇÃO DE RESULTADOS

Nesse capitulo, serão expostos os resultados alcançados utilizando os métodos de avaliação propostos. Primeiramente, na Seção 5.1, serão apresentadas as características do ambiente de testes, bem como a metodologia empregada na obtenção dos dados necessários para os experimentos. Os resultados obtidos serão expostos na Seção 5.2. Por fim, a análise comparativa e uma discussão sobre os dados serão apresentadas na Seção 5.3.

5.1 CONJUNTO DE TESTES

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 realizar a avaliação é necessário um *dataset* que represente o comportamento a ser estudado, desta forma foram elaboradas duas versões de conjuntos de dados[1] para serem utilizados nos testes, onde cada versão formou um *dataset* de comportamento normal e anormal a partir dos dados, obtidos pela captura de instâncias do Wordpress, o objeto alvo dos experimentos.

A escolha de utilizar o *Sysdig* como ferramenta para obter logs de sistema nesse trabalho representa uma abordagem alternativa em relação ao trabalho (Castanhel et al., 2020a), no qual foi optado pelo uso do *Strace*. O *Sysdig* fornece uma visão abrangente do ambiente de contêineres, com métricas detalhadas em tempo real, auditoria de eventos e informações sobre processos em execução. O *Strace*, por outro lado, se concentra em um nível mais baixo de detalhes, rastreando as chamadas de sistema feitas por processos individuais. Isso permite que seja observado exatamente quais chamadas de sistema estão sendo feitas por um processo, ajudando na depuração de problemas específicos, como bloqueios, erros de E/S ou problemas de compatibilidade.

Em um traço está contido um registro de todas as chamadas de sistema que ocorrem durante a execução de um aplicativo. No entanto, a estratégia de observação afeta quais chamadas de sistema estarão presentes nesse registro. Ao comparar as chamadas de sistema coletadas a partir de duas perspectivas diferentes, surgem notáveis diferenças. Examinando os conjuntos de dados, fica evidente uma variação significativa no número médio de chamadas observadas. No conjunto de dados do *Sysdig*, a média de chamadas de sistema para observações regulares (consideradas comportamento normal) é de 193 chamadas por registro, enquanto no conjunto de dados do *strace*, esse número é de 104 chamadas.

Essa disparidade é causada pelos diferentes formatos de representação. O *Sysdig* gera um registro de observação tanto quando uma chamada de sistema é solicitada quanto quando é concluída, resultando em duas entradas para cada chamada (Degioanni e Grasso, 2022). Para simplificar os dados, eliminamos essas entradas duplicadas, considerando apenas a observação única quando a chamada é concluída.

Para a formação do conjunto de dados utilizados nos experimentos, o processo de coleta ocorreu como descrito na Seção 4.2, desta vez utilizando como objeto alvo o Wordpress na versão 4.9.2 executando em um contêiner com as modificações mencionadas. Para esta versão, a simulação dos comportamentos normais e anormais ocorreu de forma diferente. Para a simulação de comportamento normal, foram elaboradas 10 rotinas de execução envolvendo atividades normais corriqueiras de interação com o blog, e cada rotina foi executada 10 vezes, onde o fluxo

[1] A base pode ser encontrada em `https://github.com/Carmofrasao/TCC`.

de chamadas de cada execução foi redirecionado para um arquivo diferente, totalizando 100 arquivos com fluxos de comportamento normal.

A simulação do comportamento anômalo ocorreu de forma similar, onde foram exploradas 10 vulnerabilidades diferentes:

- CVE-2019-9978 - Plugin Social Warfare (Versão: < 3.5.3). Permite a execução de código arbitrário no alvo em uma funcionalidade que gerencia a importação de configurações;
- CVE-2020-25213 – O plugin File Manager (wp-file-manager) (versão: <= 6.9). Permite o *upload* e execução código PHP arbitrário;
- Vulnerabilidade presente no plugin Simple File List (simple-file-list) (versão: < 4.2.3). *Upload* de código arbitrário e execução no contexto do processo do servidor da Web. Isso pode facilitar o acesso não autorizado ou o aumento de privilégios;
- Vulnerabilidade presente no plugin Payments forms (versão: 2.4.6). Permite a injeção de código arbitrário;
- CVE-2022-3142 – O plugin NEX-Forms (versão: < 7.9.7). Permite a injeção de SQL autenticada.
- Vulnerabilidade presente no plugin Mail Masta (versão: 1.0). Permite inclusão de arquivo, geralmente explorando um mecanismo de "inclusão dinâmica de arquivos" implementado no aplicativo de destino;
- Vulnerabilidade presente no plugin Really Simple Guest Post (versão: 1.0.6). Permite fazer *upload* de arquivos indevidos;
- CVE-2015-5065 – O plugin Paypal Currency Converter Basic For WooCommerce - File Read (versão: < 1.4). Download remoto de arquivos;
- Vulnerabilidade presente no plugin LeagueManager (versão: 3.9.11). Injeção de código SQL; e
- Vulnerabilidade presente no plugin CodeArt Google MP3 Player. Download do arquivo de configuração config.php e extração de dados de acesso do banco de dados.

Cada rotina foi executada 10 vezes e em cada execução o fluxo de chamadas foi redirecionado para um arquivo diferente, totalizando 100 arquivos representando o conjunto de dados anormais.

5.2 RESULTADOS

Para a avaliação dos resultados, dois conjuntos de testes foram feitos. Primeiramente foi avaliado o potencial em utilizar as informações coletadas pelo *sysdig* para um processo de detecção de anomalias. O segundo teste é um comparativo entre o trabalho de graduação do Gabriel Ruschel Castanhel (Castanhel et al., 2020a), onde a coleta de dados foi realizada usando a ferramenta *strace*, com a classificação das *system calls* de acordo com níveis de ameaça. (Castanhel et al., 2020a) buscou avaliar diferentes técnicas de detecção de anomalias baseadas em *system calls*, considerando o impacto do tamanho da janela deslizante e a filtragem de chamadas. O foco principal foi a análise de uma aplicação específica em contêiner, o *Wordpress*, devido às suas vulnerabilidades comuns. Através da comparação entre os dois conjunto de dados, visamos investigar o impacto ao observar *system calls* utilizando o *strace* e o *sysdig*.

5.2.1 Estratégia utilizada

A avaliação foi feita considerando a base de dados apresentada, a partir dela foi gerado um histograma para cada traço. O histograma contém o numero de chamadas de cada *system call*, em cada execução dos testes.

O conjunto de dados foi criado combinando sua base de comportamento correspondente e inclui uma etiqueta que classifica cada janela como “normal” ou “anormal”. Para fins de avaliação, o conjunto de dados foi dividido em duas partes iguais, sendo a primeira utilizada para treinamento e a segunda para realizar os testes.

Para a avaliação do desempenho de classificação dos algoritmos, foram utilizadas sete métricas: ROC, *precision*, *recall* e *f1-score*, *accuracy*, *balanced accuracy* (BAC) e *brier score*. A curva ROC é um gráfico que ilustra o desempenho de um modelo de classificador binário em valores de limiar variáveis. *Precision* representa a razão entre as detecções corretamente previstas e todas as detecções que ocorreram, onde valores altos representam baixa ocorrência de falsos-positivos. O *recall* representa a fração de detecções identificadas dentro de todas as detecções possíveis. O *f1-score* combina os valores de *precision* e *recall* em um único resultado, que indica a qualidade geral do modelo. *Accuracy* é uma métrica que avalia a desempenho geral do modelo com base em sua quantidade de acertos ao ser testado com os dados de teste. BAC é a média aritmética da sensibilidade e da especificidade, e seu caso de uso é quando se lida com dados desequilibrados, ou seja, quando uma das classes-alvo aparece muito mais do que a outra. A pontuação de *Brier score* é uma métrica de avaliação usada para verificar a qualidade de uma pontuação de probabilidade prevista. Ele é semelhante ao erro quadrático médio, mas só é aplicado a pontuações de probabilidade de previsão, cujos valores variam entre 0 e 1.

5.2.2 Detecção de Anomalias

Os resultados do primeiro experimento utilizando o *dataset* podem ser encontrados na Tabela 5.1. Foram executados 7 algoritmos nesta avaliação, são eles: Random Forest Classifier (RFC), XGBoost (XGB), Decision Tree Classifier (DTC), Nu-Support Vector Classification (NuSVC), Multi-layer Perceptron Classifier (MLPC), AdaBoost (AB) e Stochastic Gradient Descent Classifier (SGDC).

O modelo Multi-Layer Perceptron Classifier destaca-se com os valores mais altos entre os métodos avaliados. Com uma precisão de 93.33% e uma sólida taxa de *recall* de 85.71%, o Multi-Layer Perceptron Classifier é eficaz na identificação de instâncias positivas, tornando-o uma possível opção para implementação. É importante notar que a alta precisão indica que o modelo tem uma baixa taxa de falsos positivos, o que é crucial em cenários em que identificar incorretamente uma instância positiva pode ter graves consequências.

Além disso, seu ROC de 95.88% e o *Brier Score* de 10% demonstram uma capacidade de discriminação e calibração. No entanto, a sólida taxa de *recall* de 85.71% também sugere que o modelo pode minimizar a ocorrência de falsos negativos, ou seja, a perda de instâncias positivas.

Os métodos Random Forest, Nu-Support Vector Classification e AdaBoost apresentam desempenho médio dentre os algoritmos avaliados. Eles demonstram taxas de precisão e *recall* acima de 80%, indicando uma capacidade confiável de classificar positivos e negativos. 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 e BAC também estão na média, tornando-os adequados para várias aplicações. No entanto, é essencial considerar o impacto de falsos positivos e falsos negativos em cada contexto específico.

O Stochastic Gradient Descent Classifier (SGDC) exibe os menores valores em comparação com os outros métodos avaliados. Embora tenha uma alta taxa de *precision* de 90.91%, sua taxa de *recall* de 40.82% é inferior. Isso significa que o SGDC tende a gerar mais falsos negativos, o que pode ser crítico em situações em que a identificação precisa de instâncias positivas é fundamental. Além disso, seu ROC e BAC estão abaixo dos demais modelos, indicando uma capacidade de discriminação e calibração menor. O *Brier* Score mais alto também sugere que o SGD pode não ser tão bem calibrado quanto os outros métodos, tornando-o menos adequado para aplicações sensíveis a erros.

O MLP treinado com *sysdig* apresentou um *recall* de 85.71%, enquanto o correspondente treinado com *strace* atingiu 65.85%. Essa diferença sugere que a ferramenta *sysdig* desempenhou um papel crucial na captura e representação dos dados, resultando em modelos mais sensíveis na detecção de intrusões. A preferência pelo *sysdig* é respaldada pelos resultados superiores em *recall*, indicando uma capacidade aprimorada de identificar verdadeiros positivos, um aspecto crítico na eficácia de sistemas de detecção de intrusões. Assim, a utilização do *sysdig* emerge como uma abordagem mais relevante e impactante na construção de modelos de IDS eficazes.

Classificador	**ROC**	***Precision***	***Recall***	**F1**	***Accuracy***	**BAC**	***Brier***
AB	95.84%	83.67%	83.67%	83.67%	84.00%	83.99%	16.00%
DT	77.79%	84.62%	67.35%	75.00%	78.00%	77.79%	22.00%
MLP	95.88%	93.33%	85.71%	89.36%	90.00%	89.92%	10.00%
NuSV	94.14%	86.00%	87.76%	86.87%	87.00%	87.01%	13.00%
RF	90.84%	80.39%	83.63%	82.00%	82.00%	82.03%	18.00%
SGD	68.45%	90.91%	40.82%	56.34%	69.00%	68.45%	31.00%
XGB	91.20%	80.85%	77.55%	79.17%	80.00%	79.95%	20.00%

Tabela 5.1: Desempenho dos algoritmos de ML considerando todas as chamadas

5.2.3 Comparativo ponto de Observação

Visando comparar o *sysdig* e o *strace*, com o intuito em identificar diferenças entre pontos de observação distintos. Para manter as comparações justas, usamos somente parte dos dados presentes nos conjuntos de dados, pois algumas vulnerabilidades presentes nos dados do *strace* não puderam ser exploradas com o *sysdig*, e vulnerabilidades presentes nos dados do *sysdig* não foram exploradas com o *strace*. Desta forma, para o conjunto de dados do *strace* foi selecionado 6 comportamentos, 3 interações normais e 3 anormais, 5 execuções para cada comportamento, totalizando 30 arquivos de log. Já para o conjunto de dados do *sysdig* foram utilizados 6 comportamentos, 3 interações normais e 3 anormais, 10 execuções cada totalizando 60 arquivos de log. Os testes realizados visam identificar as discrepâncias presentes nas coletas de cada ferramenta, visando identificar chamadas de sistemas especificas que estão possivelmente relacionadas com ataques. Além de buscar por padrões entre os dois conjunto para identificar qual a viabilidade do uso de cada ferramenta.

Ao examinar o conjunto de operações que não sinalizam intenções maliciosas, os dados obtidos do *strace* revelam um maior número de chamadas de sistema associadas à comunicação, como *sendto* e *recvfrom*, e também operações de manipulação do sistema operativo, tais como *poll*, quando comparadas ao *sysdig*. Por outro lado, no caso do *sysdig*, é notável um aumento nas interações que fazem uso de operações temporais (com chamadas de sistema como *setitimer*), chamadas de leitura, gestão de processos (como *access*) e manipulação de dispositivos (com chamadas de sistema como *select*), como indicado na Figura 5.2.

Essas foram as principais discrepâncias observadas em relação a aplicações não maliciosas. É importante ressaltar que as chamadas de sistema em grupos de funcionalidades específicas foram mais frequentes dependendo da abordagem de observação e monitoramento adotada (Heinrich et al., 2023). Isso estava de acordo com nossa suposição inicial, considerando que esperávamos diferenças nas chamadas observadas em cada perspectiva. A Figura 5.1 descreve o ambiente virtual avaliando, onde identificamos influencia da estratégia de coleta.

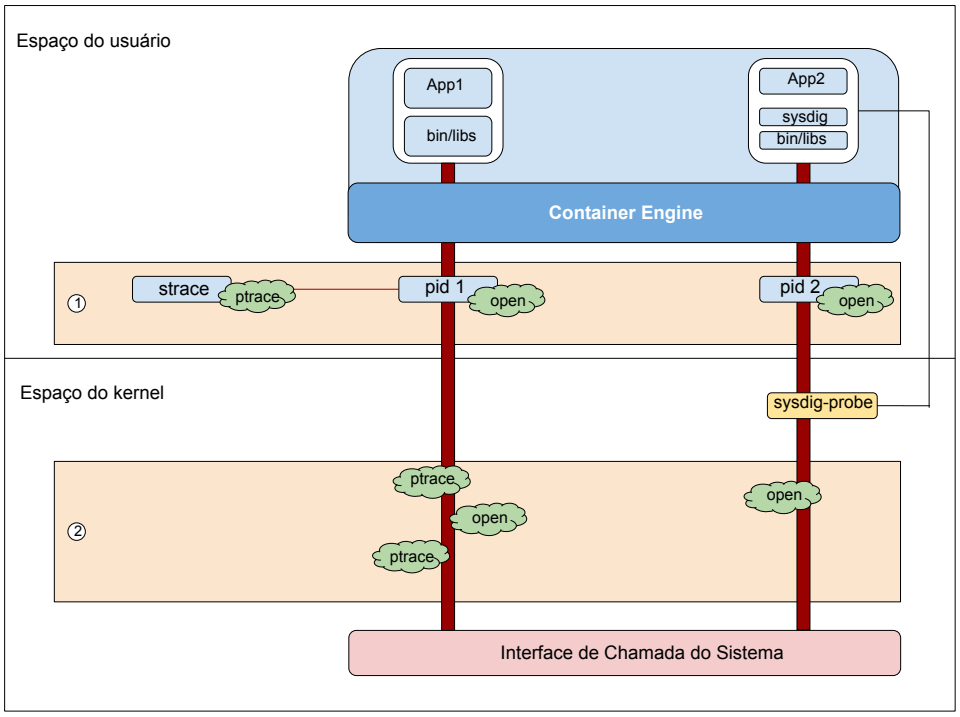


Figura 5.1: Comparação entre a perspetiva de dados *strace* e *sysdig*.

Quando se trata de amostras maliciosas, a distribuição das chamadas de sistema entre as duas perspectivas permaneceu a mesma, como mostra a Figura 5.3, mas houve valores anômalos específicos que desempenharam um papel crucial na detecção de anomalias. Em relação às chamadas com maior número de ocorrências em ambos os conjuntos de dados maliciosos, eles são apresentados na Figura 5.4. É fundamental observar que escolhemos analisar as chamadas de sistema em ambos os conjuntos de dados que tiveram pelo menos cinquenta interações, um critério que escolhemos pois este valor representa o patamar que abrange 96% do grupo de chamadas de sistema com comportamento anômalo.

Além disso, em adição aos grupos de chamadas de sistema observados no conjunto não malicioso, observamos variações na frequência de operações como *getcwd* e *open* nas atividades realizadas por aplicativos maliciosos. Essa variação é esperada quando se analisam aplicativos em busca de anomalias.

5.3 DISCUSSÃO

A escolha de utilizar o *Sysdig* em vez do *Strace* para coletar logs do sistema pode ser vantajosa pois fornece uma visão mais abrangente do ambiente de contêineres, capturando métricas detalhadas em tempo real, análises de tráfego de rede e informações sobre processos em execução. Isso nos permite monitorar o sistema como um todo, identificando problemas de desempenho, segurança ou conformidade.

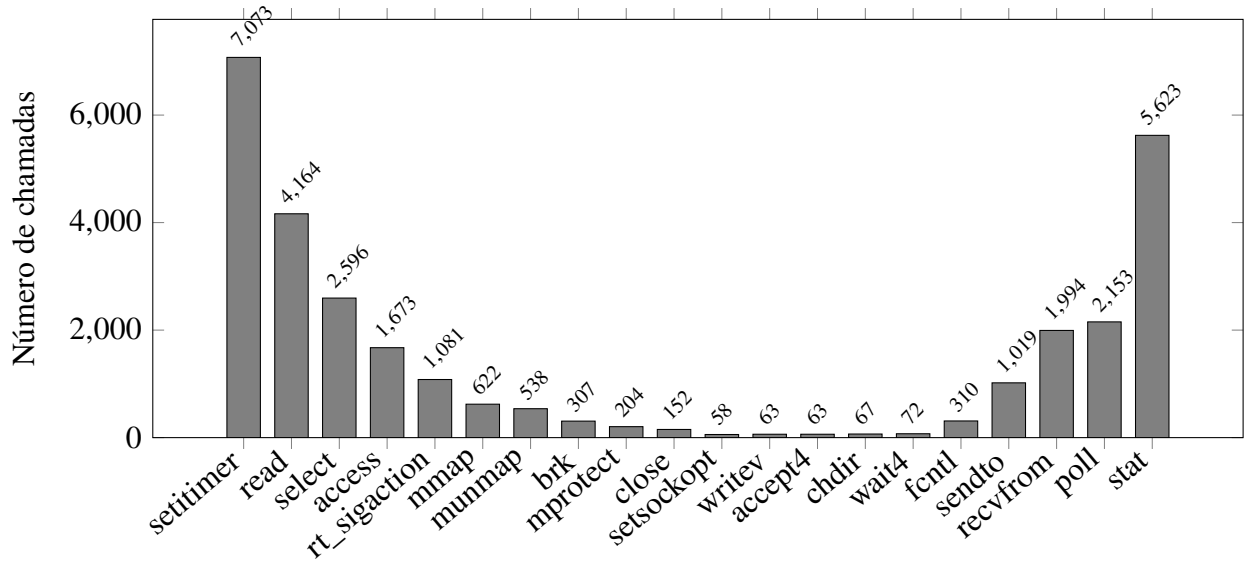


Figura 5.2: Diferença de Chamadas de Sistema Sysdig vs Strace (normal)

A pesquisa avaliou vários algoritmos de classificação, com destaque para o *Multi-Layer Perceptron Classifier* (MLPC) devido a sua precisão de 93.33% e um *recall* de 85.71%. O MLPC também demonstrou boa discriminação e calibração, com um ROC de 95.88% e um baixo *Brier Score* de 10%. Outros algoritmos, como *Random Forest*, *Nu-Support Vector Classification* e *AdaBoost*, também apresentaram desempenho sólido com precisão e *recall* acima de 80%. No entanto, o *Stochastic Gradient Descent Classifier* (SGDC) teve um desempenho inferior, com *recall* de apenas 40.82% e um *Brier Score* mais alto.

Nossas observações destacaram diferenças nas chamadas de sistema entre as perspectivas do *Sysdig* e *Strace*, tanto para operações não maliciosas quanto maliciosas. Isso é esperado, considerando as abordagens de observação distintas. Com nossos dados, obtivemos bons resultados nas avaliações feitas, mostrando a possibilidade de uso na pratica, assim como os dados obtidos com o *Strace*. Assim, podemos concluir quem ambas as ferramentas são uteis passiveis de serem utilizadas em IDSs.

Este estudo contribui para o campo da detecção de anomalias em sistemas de contêineres, fornecendo percepções sobre a eficácia de diferentes algoritmos e métodos de coleta de dados. Nossos resultados sugerem que o MLP é uma escolha promissora para a detecção de anomalias em sistemas de contêineres. Além disso, destacamos a importância da consistência na observação e da compreensão das variações nas chamadas de sistema ao lidar com a detecção de anomalias.

Contudo, reconhecemos algumas limitações em nosso estudo, como o tamanho do conjunto de dados e a seleção de algoritmos. Essas limitações podem afetar a generalização de nossos resultados. Pesquisas futuras podem se concentrar em expandir o conjunto de dados e explorar algoritmos adicionais.

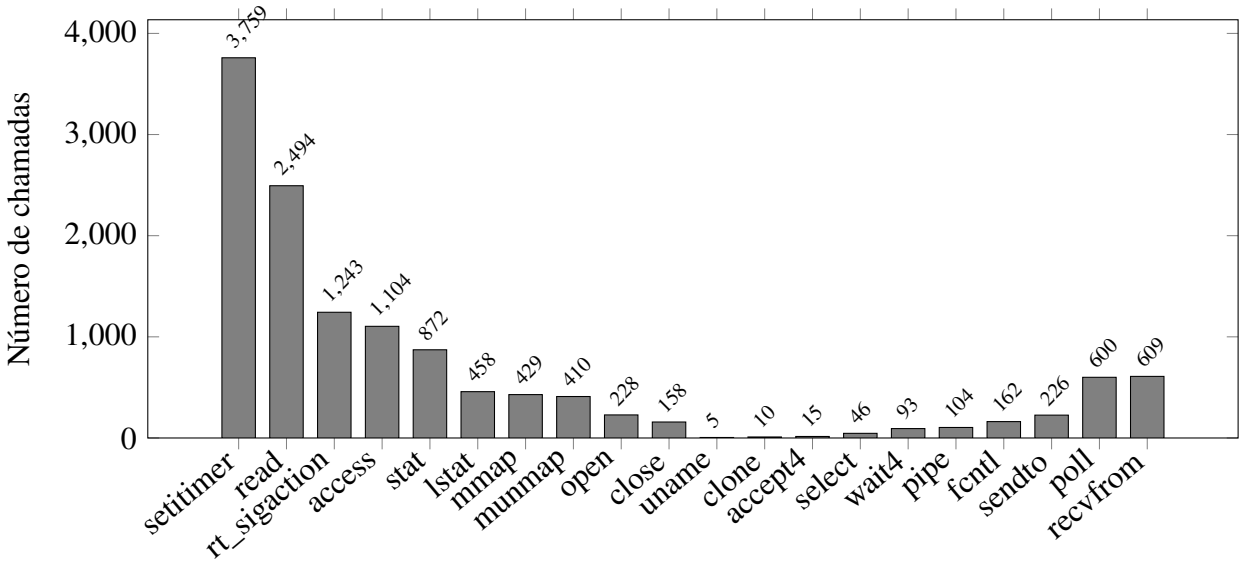


Figura 5.3: Diferença de Chamadas de Sistema Sysdig vs Strace (anormal)

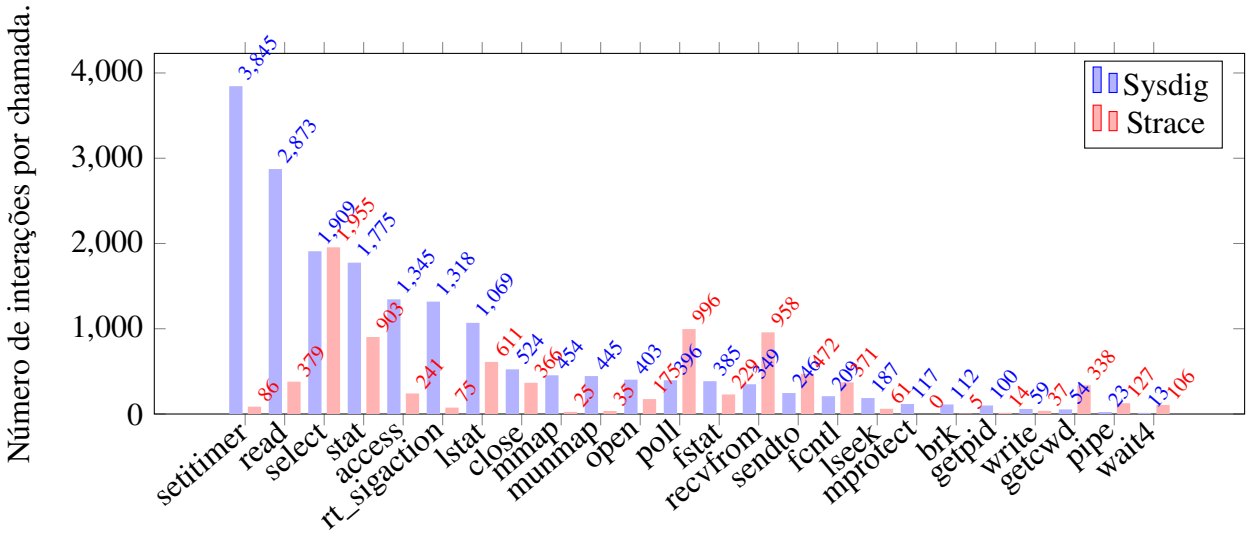


Figura 5.4: As chamadas de sistema com 50 ou mais requisições usadas por interações mal-intencionadas. A comparação entre o *sysdig* e o *strace* demonstra a diferença entre o número de interações realizadas por aplicativos mal-intencionados

6 CONCLUSÃO

Neste trabalho de conclusão de curso foi estudada a possibilidade e viabilidade da detecção de intrusão por anomalia com foco em um ambiente de contêiner, utilizando *system calls* para definir o comportamento de um contêiner alvo, onde foi possível analisar o desempenho de diferentes algoritmos de ML, e também comparar a diferença de resultados dos métodos quanto ao tipo de *system calls* levadas em consideração para treinar e testar os classificadores. Além disso, foi feita a comparação entra o modelo de coleta de dados *strace* e o *sysdig*, e como essa escolha afeta no treinamento dos algoritmos de ML.

Foi realizada a coleta de dados da classe normal e anômala para formar o *dataset* utilizando uma versão do Wordpress, o objeto alvo, executando em um contêiner. Os dados foram coletados de maneira estruturada e contou com 10 exemplares de ataques diferentes ao objeto alvo. A partir dos dados coletados, foi formado um histograma para o treinamento e teste de alguns algoritmos de ML. Também foram aplicados diferentes filtros no conjunto de dados gerado a partir do *strace* e no conjuntos gerado pelo *sysdig*, para fazer as comparações necessárias, tais como maior incidência de chamadas, diferença entre chamadas especificas, entre outras comparações que são expostas no texto.

Na avaliação dos resultados obtidos pelos classificadores, o modelo *Multi-Layer Perceptron Classifier* se mostrou mais eficiente, com uma alta precisão de 93.33% e uma sólida taxa de *recall* de 85.71%, o MLPC é eficaz na identificação de instâncias positivas, sendo uma opção para implementação. Além disso, seu ROC excepcionalmente alto de 95.88% e o *Brier Score* muito baixo de 10% demonstram uma capacidade notável de discriminação e calibração.

Foi observado que é viável monitorar o comportamento de um contêiner através da análise das *system calls* feitas por ele, o que se mostrou ser uma ferramenta valiosa para identificar anomalias neste cenário. Essa área de pesquisa tem ganhado popularidade nos últimos anos, com estudos que empregam diversas abordagens tanto na detecção de anomalias quanto na coleta e tratamento dos dados provenientes do contêiner, produzindo resultados significativos. No entanto, ainda é desafiador encontrar trabalhos que compartilhem os conjuntos de dados utilizados.

Ao comparar diferentes estratégias de observação foi observado uma diferença nas chamadas de sistema predominantes entre as duas perspectivas, como por exemplo *sendto* e *poll* para o *Strace* e para o *Sysdig*, as chamadas *setitimer* e *select* tiveram mais aparições nos dados coletados, destacando a importância da escolha da ferramenta de observação e monitoramento.

REFERÊNCIAS

Abed, A. S., Clancy, C. e Levy, D. S. (2015a). Intrusion detection system for applications using linux containers. Em *Security and Trust Management*, páginas 123–135. Springer.

Abed, A. S., Clancy, T. C. e Levy, D. S. (2015b). Applying bag of system calls for anomalous behavior detection of applications in linux containers. Em *2015 IEEE globecom workshops (GC Wkshps)*, páginas 1–5. IEEE.

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

Amazon (2023). Integridade de instâncias do contêiner. `https://docs.aws.amazon.com/pt_br/AmazonECS/latest/developerguide/container-instance-health.html`.

Bridges, R. A., Glass-Vanderlan, T. R., Iannacone, M. D., Vincent, M. S. e Chen, Q. (2019). A survey of intrusion detection systems leveraging host data. *ACM Computing Surveys (CSUR)*, 52(6):1–35.

Carbonell, J. G., Michalski, R. S. e Mitchell, T. M. (1983). An overview of machine learning. *Machine learning*, páginas 3–23.

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

Castanhel, G. R., Heinrich, T., Ceschin, F. e Maziero, C. A. (2020a). Detecçao de anomalias: Estudo de técnicas de identificaçao de ataques em um ambiente de contêiner. Em *Anais Estendidos do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, páginas 169–182. SBC.

Castanhel, G. R., Heinrich, T., Ceschin, F. e Maziero, C. A. (2020b). Sliding window: The impact of trace size in anomaly detection system for containers through machine learning. Em *Anais da XVIII Escola Regional de Redes de Computadores*, páginas 141–146. SBC.

Ceschin, F., Gomes, H. M., Botacin, M., Bifet, A., Pfahringer, B., Oliveira, L. S. e Grégio, A. (2020). Machine learning (in) security: A stream of problems. *arXiv preprint arXiv:2010.16045*.

Chandola, V., Banerjee, A. e Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

Degioanni, L. e Grasso, L. (2022). Practical cloud native security with falco. O'Reilly Media.

Docker (2023). Use containers to build, share and run your applications. `https://www.docker.com/resources/what-container/`.

Du, Q., Xie, T. e He, Y. (2018). Anomaly detection and diagnosis for container-based microservices with performance monitoring. Em *Algorithms and Architectures for Parallel Processing: 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part IV 18*, páginas 560–572. Springer.

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

Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. *International Conference on Machine Learning (ICML-2000)*.

Flora, J. e Antunes, N. (2019). Studying the applicability of intrusion detection to multi-tenant container environments. Em *2019 15th European Dependable Computing Conference (EDCC)*, páginas 133–136. IEEE.

Heinrich, T., Will, N. C., Obelheiro, R. R., Maziero, C. A. e Vizinhos-PR-Brasil, D. (2023). Uso de chamadas wasi para a identificação de ameaças em aplicações webassembly. *Anais Estendidos do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. SBC*.

Hodge, V. e Austin, J. (2004). A survey of outlier detection methodologies. *Artificial intelligence review*, 22:85–126.

Joy, A. M. (2015). Performance comparison between linux containers and virtual machines. Em *2015 international conference on advances in computer engineering and applications*, páginas 342–346. IEEE.

Kubernetes (2023). Kubernetes components. `https://kubernetes.io/docs/concepts/overview/components/`.

Lakhina, A., Crovella, M. e Diot, C. (2004). Diagnosing network-wide traffic anomalies. *ACM SIGCOMM computer communication review*, 34(4):219–230.

Laureano, M. A. P., Maziero, C. A. e Jamhour, E. (2003). Detecção de intrusão em máquinas virtuais. *5º Simpósio de Segurança em Informática–SSI. São José dos Campos*, páginas 1–7.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C. e Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24.

Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K. e Zhou, Q. (2018). A measurement study on linux container security: Attacks and countermeasures. Em *Proceedings of the 34th Annual Computer Security Applications Conference*, páginas 418–429.

Liu, M., Xue, Z., Xu, X., Zhong, C. e Chen, J. (2018). Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys (CSUR)*, 51(5):1–36.

Lopez, M., Figueiredo, U., Lobato, A. e Duarte, O. C. (2014). Broflow: Um sistema eficiente de detecção e prevenção de intrusão em redes definidas por software. Em *Anais do XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, páginas 108–121. SBC.

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

Merkel, D. et al. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2.

Mohri, M., Rostamizadeh, A. e Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

Nasteski, V. (2017). An overview of the supervised machine learning methods. *Horizons. b*, 4:51–62.

RedHat (2023a). Docker: desenvolvimento de aplicações em containers. `https://www.redhat.com/pt-br/topics/containers/what-is-docker`.

RedHat (2023b). Kubernetes. `https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes`.

Rocha, S. L., Nze, G. D. A. e 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. Em *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, páginas 1–4. IEEE.

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

Shen, J., Zeng, F., Zhang, W., Tao, Y. e Tao, S. (2022). A clustered learning framework for host based intrusion detection in container environment. Em *2022 IEEE International Conference on Communications Workshops (ICC Workshops)*, páginas 409–414. IEEE.

Srinivasan, S., Kumar, A., Mahajan, M., Sitaram, D. e Gupta, S. (2019). Probabilistic real-time intrusion detection system for docker containers. Em *Security in Computing and Communications: 6th International Symposium, SSCC 2018, Bangalore, India, September 19–22, 2018, Revised Selected Papers 6*, páginas 336–347. Springer.

Sturm, R., Pollard, C. e 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. e Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead. *IEEE access*, 7:52976–52996.

Sysdig (2014). Sysdig vs dtrace vs strace: A technical discussion. `https://sysdig.com/blog/sysdig-vs-dtrace-vs-strace-a-technical-discussion/`.

Sysdig (2017). Visão geral do sysdig. `https://github.com/draios/sysdig/wiki/Sysdig-Overview`.

Sysdig (2019). Sysdig e falco agora alimentados por ebpf. `https://sysdig.com/blog/sysdig-and-falco-now-powered-by-ebpf/`.

Thottan, M. e Ji, C. (2003). Anomaly detection in ip networks. *IEEE Transactions on signal processing*, 51(8):2191–2204.

Vinchurkar, D. P. e Reshamwala, A. (2012). A review of intrusion detection system using neural network and machine learning. *J. Eng. Sci. Innov. Technol*, 1:54–63.

Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T. e De Rose, C. A. (2013). Performance evaluation of container-based virtualization for high performance computing environments. Em *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, páginas 233–240. IEEE.

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