Dando uma espiada: uma avaliação da anomalia Detecção usando chamadas do sistema para contêineres

Gabriel R. Castanhel, Tiago Heinrich, Fabr´ÿcio Ceschin, Carlos Maziero Programa de Pós-Graduação em Ciência da Computação Universidade Federal do Paraná Curitiba, Brasil, 81530–015 E-mail: {grc15,theinrich,fjoceschin,maziero}@inf.ufpr.br

Resumo—O crescimento do uso da virtualização nos últimos dez anos tem contribuído para o aprimoramento desta tecnologia. A prática de implantação e gerenciamento desse tipo de ambiente isolado gera dúvidas quanto à segurança desses sistemas. Considerando a proximidade do host a um contêiner, as abordagens que usam sistemas de detecção de anomalias tentam monitorar e detectar comportamentos inesperados. Nosso trabalho visa usar chamadas de sistema para identificar ameaças dentro de um ambiente de contêiner, usando estratégias baseadas em aprendizado de máquina para distinguir entre comportamentos esperados e inesperados (possíveis ameaças).

Termos do Índice—Detecção de Intrusão, Segurança de Computadores, Contêineres

I. INTRODUÇÃO

A virtualização no nível do sistema operacional, também conhecida como conteinerização, é um tipo de virtualização na qual vários espaços de usuário isolados são fornecidos por um único kernel. Cada espaço de usuário, denominado container, executa uma aplicação com suas dependências e recursos, isolada das demais aplicações rodando no mesmo host físico. Isso permite que vários contêineres com diferentes estruturas e aplicativos sejam executados lado a lado, compartilhando o mesmo kernel do sistema operacional.

Atualmente, a popularidade deste tipo de virtualização aumenta as preocupações de segurança, devido à proximidade entre as aplicações executadas em um container e o host, pois a camada de containerização é muito mais fina do que uma pilha de virtualização completa. Vários ataques foram encontrados e explorados nos últimos anos, como execução remota de código, escalonamento de privilégios, adulteração, entre outros [1].

A detecção de intrusão é uma maneira de identificar e prevenir atividades maliciosas em um sistema. Os recursos utilizados para tais atividades são o Sistema de Detecção de Intrusão (IDS) e o Sistema de Prevenção de Intrusão (IPS) [2]. Especificamente, um IDS realiza a identificação de intrusão usando diferentes técnicas que podem ser baseadas em assinaturas, que realizam a comparação de assinaturas com uma base conhecida de ameaças, ou baseadas em anomalias, nas quais o comportamento normal do sistema é previamente conhecido e os desvios dele são classificados como ameaças [3].

Olhando apenas para contêineres de segmentação de detecção de intrusão, algumas diferenças podem ser apontadas. Os contêineres são mais leves do que as máquinas virtuais e, portanto, mais fáceis e rápidos de gerenciar. Além disso, a lacuna semântica entre o ambiente virtual e o host é muito pequena em contêineres, em comparação com as máquinas virtuais tradicionais [4]. Isso permite um observador externo para coletar informações muito detalhadas sobre os aplicativos em execução dentro de um contêiner.

Um invasor pode explorar diferentes técnicas para comprometer, obter acesso ou até mesmo executar código em um ambiente de contêiner. Tais ataques podem ser realizados a partir de uma imagem comprometida, executando aplicativos com permissões desnecessárias (por exemplo, como root), explorando uma vulnerabilidade no aplicativo que executa o contêiner ou um aplicativo com um ambiente mal configurado [1].

Este trabalho estuda e compara a eficácia de um conjunto de métodos na identificação de atividades maliciosas em um container. Um observador externo ao container coleta informações detalhadas sobre sua execução, na forma de uma sequência de chamadas de sistema emitidas pela aplicação. Esses dados são usados para treinar classificadores para construir um modelo de “comportamento normal” de cada contêiner e, consequentemente, permitir identificar anomalias.

Em resumo, este artigo traz as seguintes contribuições: • Uma discussão sobre o uso de detecção de intrusão usando chamadas de sistema em contêineres e como foi implementado; • Uma metodologia para a identificação de ataques dentro de um container utilizando aprendizado de máquina, alcançando altos índices de acurácia e precisão para ambas as estratégias apresentadas; • Exploramos o impacto do tamanho da janela e da filtragem de chamadas do sistema para um sistema de detecção de intrusão em um container; • Um conjunto de dados que permite a avaliação do uso do siste chamadas para detectar ameaças dentro de um contêiner.

O resumo deste artigo está estruturado da seguinte forma: a Seção II apresenta os trabalhos relacionados; A Seção III apresenta os antecedentes; A Seção IV discute a proposta do artigo; A Seção V apresenta a avaliação e a Seção VI conclui o trabalho.

II. TRABALHO RELACIONADO

Do ponto de vista do invasor, existem várias possibilidades de ataque a um sistema de virtualização, como explorar a virtualização para extrair informações privadas dos usuários, iniciar ataques Distributed Denial of Service (DDoS) ou escalar a invasão para várias instâncias de VM. A literatura destaca estudos voltados para o monitoramento da virtualização em diferentes níveis, a fim de identificar tais ações maliciosas [5, 6].

A detecção de intrusão em ambientes de virtualização traz alguns desafios. Se o IDS for colocado dentro do convidado, ele terá uma visão avançada do aplicativo monitorado, mas também ficará exposto a ataques direcionados a ele. Por outro lado, se o IDS reside

978-1-6654-2744-9/21/$31,00

fora do convidado monitorado, está protegido contra tais ataques, mas sua visão da execução do aplicativo convidado é muito mais pobre, devido à lacuna semântica [4].

No entanto, ao usar a virtualização baseada em contêiner, a lacuna semântica é bastante reduzida, porque todos os contêineres compartilham o mesmo kernel. Assim, um IDS rodando diretamente no sistema operacional host é capaz de observar completamente o comportamento dos processos rodando em um container [4].

Um dos primeiros estudos que explorou o campo de detecção de intrusão baseada em chamada de sistema foi apresentado por [7]. Propôs um método inspirado nos mecanismos e algoritmos utilizados pelos sistemas imunológicos naturais. O estudo observou que sequências curtas de chamadas de sistema pareciam manter uma consistência notável entre os muitos conjuntos possíveis de chamadas de sistema dos possíveis caminhos de execução de um programa.

Isso inspirou o uso de sequências curtas de chamadas de sistema para definir o comportamento normal do sistema e apresentou uma maneira simples e eficiente de detectar anomalias, com possíveis aplicações em cenários de tempo real.

Para abordagens baseadas em virtualização de containers, pode-se destacar [8], que apresenta um modelo para identificação de anomalias em aplicações rodando dentro do Docker. A estratégia é usar n-gramas para identificar a probabilidade de um evento ocorrer. O experimento atinge uma precisão de até 97% para o conjunto de dados UNM [9]. No entanto, esse conjunto de dados é muito antigo e não representa os aplicativos e ambientes virtuais atuais.

Ainda no contexto de containers, [10] apresenta uma análise preliminar de viabilidade para detecção de intrusão baseada em anomalias, com foco nas tecnologias Docker e LXC. O artigo propõe uma arquitetura de análise e captura para chamadas de sistema, a aplicação dos algoritmos Sequence Time-Delay Embedding (STIDE) e Bag of System Calls (BoSC), e estuda o processo de treinamento em diferentes casos. O estudo treinou os dois algoritmos com tamanhos de janela variando de 3 a 6 chamadas de sistema e calculou a inclinação da curva de crescimento, que significa a taxa de novas janelas adicionadas à base de comportamento normal de cada classificador após um período de tempo. Os resultados destacam um estado de aprendizado estável para STIDE com janelas de tamanho 3 e 4 e de 3 a 6 para BoSC, o que significa que a melhor configuração obtida foi usando janelas de tamanho 3 e 4. O banco de dados usado para validação e experimentação foi desenvolvido para o estudo, mas não está disponível publicamente.

O estudo de [11], centra-se na deteção de intrusão por anomalias em ambientes de contentores, aplicando uma técnica que combina o BoSC com a técnica do STIDE. A análise do comportamento do container é feita após seu fechamento, com o auxílio de uma tabela contendo todas as chamadas de sistema distintas com o respectivo número total de ocorrências. O método lê o fluxo de chamadas do sistema por épocas e desliza uma janela de tamanho 10 por cada época produzindo um BoSC para cada janela, que é usado para detectar anomalias, que por sua vez é declarada se o número de disparidades na base de comportamento normal excede um limite definido. O classificador alcançou uma taxa de detecção de 100% e uma taxa de falso positivo de 0,58% para época de tamanho 5.000 e limite de detecção de 10% o tamanho da época. O banco de dados do experimento não está disponível.

III. FUNDO

Esta Seção apresenta os principais conceitos para a compreensão do trabalho, discutindo pontos como detecção de anomalias, chamadas de sistema para IDS e virtualização conteinerizada.

A. Chamadas do sistema

Chamadas de sistema são mecanismos disponíveis para a interação entre uma aplicação (ou processo) e o kernel do sistema operacional. Quando um programa precisa executar operações privilegiadas, as solicitações são feitas por meio de chamadas do sistema, geralmente porque os processos de nível de usuário não têm permissão para executar tais operações.
Isso ocorre para uma variedade de tarefas, como interação com recursos de hardware, gerenciamento de memória, processamento de entrada/saída, operações de rede, operações de arquivo e outros [12].

Algumas dessas ações não devem estar disponíveis para todos os processos de nível de usuário, como desligamento, acesso a áreas de memória de outros processos, alteração de IDs de usuário e desvio de políticas de controle de acesso. O kernel do sistema operacional, portanto, implementa políticas de segurança que determinam quais chamadas de sistema podem ser chamadas por quais processos de nível de usuário.

Devido à posição em que as chamadas do sistema são implementadas, sua observação fornece informações valiosas sobre as atividades executadas pelos processos no nível do usuário. O conjunto de chamadas de sistema encontradas em um sistema está diretamente relacionado ao Sistema Operacional (SO) e à arquitetura utilizada. Ferramentas como strace e ftrace [13, 14], permitem mostrar a sequência de todas as chamadas de sistema usadas por um comando ou processo em execução.

Independentemente das abordagens usadas para executar códigos maliciosos em um sistema, eles geralmente exploram a interface de chamadas do sistema para executar operações maliciosas [15]. É somente por meio dessa interface que um aplicativo comprometido interage com os serviços e recursos do sistema. O monitoramento de chamadas de sistema é uma técnica amplamente utilizada que faz uso deste recurso em comum para detectar comportamentos suspeitos de uma aplicação que possa ter sido comprometida, para que seja possível uma contramedida para minimizar o problema [16].

B. Contêineres

Embora o conceito de container tenha se popularizado apenas nos últimos anos, esta é uma técnica introduzida já na década de 1980, para realizar o “isolamento” de software usando a ferramenta chroot em sistemas Linux. Atualmente é conhecida como uma técnica de virtualização de aplicações para as mais diversas finalidades, geralmente associada a ferramentas populares como o docker.
O container é um ambiente virtual que roda em um único SO e permite o carregamento e execução de uma aplicação específica e suas dependências contidas em uma virtualização de um sistema operacional [17]. Dessa forma, o container reúne os componentes necessários para a execução da aplicação, que inclui código, bibliotecas e variáveis de ambiente, enquanto o SO hospedeiro gerencia o acesso aos recursos de hardware como memória e processador, e todas as operações necessárias do kernel.

Os contêineres promovem isolamento entre host e convidados, mas esse isolamento não é tão forte quanto na virtualização completa, onde um

o hipervisor é responsável por gerenciar e executar os sistemas convidados. O hipervisor é a camada intermediária entre as máquinas virtuais e o host, responsável pela conversão de instruções e alocação de recursos [18].

O forte isolamento fornecido pelo hipervisor tem um custo, em termos de processamento, uso de memória e facilidade de compartilhamento de recursos. Tais restrições são muito mais leves quando se utilizam containers [19]. Como os contêineres compartilham o mesmo kernel, uma única instância do sistema operacional é capaz de suportar vários contêineres isolados. Além disso, o SO virtualizado no container é mais leve que uma máquina virtual, possuindo apenas os recursos necessários para a execução da aplicação [20].

4. PROPOSTA

O monitoramento de vulnerabilidades no nível da aplicação é uma tarefa que consome o processador, sendo inviável realizar um monitoramento de dentro do container devido às suas restrições e recursos limitados, bem como ao risco de ter o IDS exposto ao invasor [21] . Portanto, uma opção seria monitorar os processos convidados do sistema hospedeiro, usando como dados as chamadas de sistema por eles geradas.

Nesse contexto, [7] apresenta um método baseado em janela de monitoramento de chamadas do sistema, que é simples e eficaz para detecção em tempo real. Abordagens usando tabelas de frequência [21, 22], algoritmos de Machine Learning (ML) [23, 24] e cadeias de Markov [25] também mostraram bons resultados em relação às taxas de detecção.

A fim de comparar diferentes técnicas possíveis para detecção de anomalias baseadas em chamadas de sistema, a abordagem proposta foca em uma aplicação rodando em um ambiente container e sendo observada do host. Nosso objetivo é explorar técnicas de ML para identificar ataques e entender como os sistemas de detecção de intrusão se comportarão com essa perspectiva externa. Como utilizamos uma técnica de janela deslizante, focamos em avaliar como o tamanho da janela vai impactar nos resultados, e fazemos uma avaliação minuciosa do tamanho correto a ser utilizado.

A Figura 1 apresenta três abordagens para coletar informações de chamada do sistema de processos convidados em execução dentro de um contêiner. A primeira estratégia é executar o IDS no nível do host, fora do contêiner (representado por App1 na Figura 1). Isso protege o IDS contra um contêiner comprometido [26].

A segunda abordagem consiste em utilizar a aplicação trace dentro do mesmo container em que a aplicação está sendo executada (representada por App2 na Figura 1), o que pode ser prejudicial ao IDS. A terceira abordagem define um contêiner privilegiado para executar o IDS (representado por App3 e App4 na Figura 1).
O container IDS deve ter direitos de acesso aos processos que estão sendo monitorados, para coletar as chamadas de sistema que eles emitem. Essa abordagem é interessante, pois permite definir um contêiner IDS completo, que pode ser facilmente implantado em outro lugar.

Este trabalho utilizou a primeira abordagem para coletar dados, usando a ferramenta strace para coletar todas as interações entre o processo convidado e o kernel. Coletamos informações suficientes para construir uma base de comportamentos normais, contendo sequências de chamadas de sistema que representam o funcionamento normal do

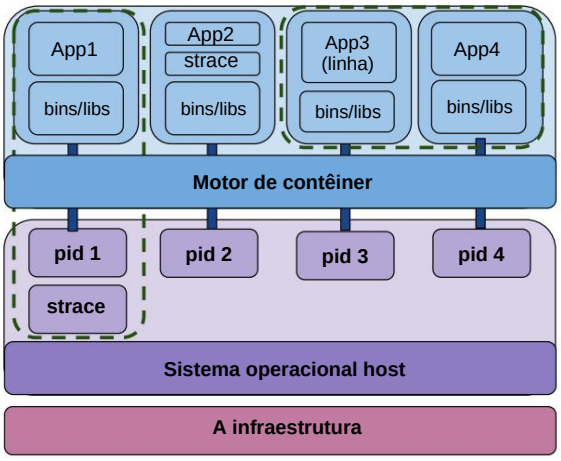


Fig. 1: Abordagens para monitorar um aplicativo convidado.

aplicação, e também sequências que a representem sob ataque ou comportamento malicioso, para avaliar a proposta.

V. AVALIAÇÃO

Construímos um protótipo de nossa proposta, para verificar se o monitoramento de uma aplicação guest de fora do container é eficaz, e também para avaliar os algoritmos utilizados para detectar anomalias no fluxo de chamadas de sistema emitidas por uma aplicação. Um conjunto de dados rotulados de comportamento normal e ataques foi
também gerado e está publicamente disponível1 .

Os experimentos foram executados em um host Linux usando o Docker como ambiente virtual. A aplicação selecionada para os testes foi o Wordpress, uma aplicação web que roda sobre o servidor web Apache. Esta escolha deve-se à sua popularidade entre aplicações web, e devido à existência de um vasto conjunto de vulnerabilidades. O uso extensivo de plugins de terceiros e temas personalizados abre as portas para falhas de segurança envolvendo Cross-site scripting (XSS), injeção de SQL, Remote Code Execution (RCE), entre outros.

A configuração da aplicação foi ajustada para reduzir a quantidade de threads e processos filhos bifurcados pelo servidor Apache, de forma a ter um fluxo mais limpo e consistente de chamadas de sistema emitidas pela aplicação. as informações de syscall coletadas contêm seus nomes, parâmetros e valores de retorno.

O conjunto de dados construído contém cinquenta traços, metade de comportamento normal que consiste em interações esperadas com o aplicativo (cinco interações diferentes) e a outra metade comportamento anômalo que consiste em ataques (cinco ataques diferentes com foco em XSS e RCE). São eles: WordPress store XSS via wp-admin [27], importação/exportação de CSV permitindo a execução de código PHP não autenticado, gerenciador de arquivos que permite o upload e a execução de código PHP [28], ignora extensões para fazer upload de shells PHP [29], e upload de arquivos de usuários não autenticados.

1O conjunto de dados e o código usado nos experimentos podem ser encontrados em https://github.com/gabrielruschel/hids-docker

Foram desenvolvidos dois grupos de testes: no primeiro caso foram utilizadas todas as chamadas do sistema, sem aplicar nenhum tipo de filtragem.
Para o segundo caso, as chamadas do sistema foram classificadas em níveis de ameaça, e as menos perigosas foram descartadas.

Para o processamento dos dados, foi utilizada uma abordagem de janela deslizante, reconstruindo o traço com uma janela de tamanho n. No total, sete diferentes tamanhos de janela foram testados usando quatro algoritmos, sendo eles K-Nearest Neighbors (KNN) com ak = 3, Random Forest (RF), Multilayer Perceptron (MLP) e AdaBoost (AB), todos eles com a parâmetros padrão do scikit-learn [30]. Para a fase de treinamento, o conjunto de dados foi dividido em 50/50 para treinamento/teste, e dez execuções foram executadas para cada classificador, alterando a semente aleatória usada na fase de divisão para gerar conjuntos diferentes, para evitar problemas de overfitting.

A eficiência de cada algoritmo foi avaliada usando quatro métricas: precisão, recall, f1-score e exatidão. A precisão representa a razão entre as detecções previstas corretamente e todas as detecções que ocorreram, onde valores altos representam uma baixa ocorrência de falsos positivos. A rechamada, por outro lado, representa a fração de detecções identificadas dentro de todas as detecções possíveis. F1-score combina os valores de precisão e recuperação em um único resultado, o que indica a qualidade geral do modelo.

As seções VA e VB discutem os experimentos realizados e os resultados obtidos, e a seção VC apresenta uma avaliação do tamanho da janela, considerando o impacto para detecção de intrusão.

A. Usando todas as chamadas do sistema

O primeiro conjunto de experimentos consiste em utilizar todas as chamadas de sistema emitidas pela aplicação. Cada chamada do sistema é representada por um identificador numérico exclusivo. A Tabela I apresenta os resultados obtidos com dez execuções, sem filtragem. Em geral os resultados parecem adequados, com precisão e exatidão acima de 90% para todos os tamanhos de janela e classificadores. A diferença entre os classificadores é bem pequena, com um f1-score e um número de recall não tendo uma diferença relevante entre cada algoritmo.

Observando apenas o tamanho da janela da Tabela I, alguns pontos podem ser destacados: (1) as janelas dos tamanhos 3 e 5 apresentam os valores mais “instáveis” em relação aos demais tamanhos de janela; (2) a diferença entre os resultados é pequena para tamanhos de janela entre 7 e 15, o que indica que elas podem apresentar um bom trade-off entre tempo de detecção e desempenho de classificação; e (3) Random Forest e AdaBoost apresentam os melhores resultados gerais.

B. Usando chamadas de sistema filtradas

O trabalho [31] propôs uma classificação de segurança para chamadas de sistema, na qual cada syscall é rotulada com um nível de ameaça, significando o quão perigosa aquela chamada de sistema pode ser para o sistema se mal utilizada. Os níveis propostos variam de 1 a 4, onde o nível 1 representa as chamadas do sistema que permitem o controle completo do sistema; o nível 2 representa as chamadas do sistema que podem ser usadas para ataques de negação de serviço (DoS); chamadas de sistema de nível 3 podem ser usadas para subverter o processo responsável; finalmente, sistema de nível 4

TABELA I: Dez execuções considerando todas as chamadas (sem filtro).

Métrica do Classificador	Tamanho da Janela 3	5	7	9	11	13	15
KNN	precisão 84,5 90,4 90,3 88,5 91,8 90,4 87,9 rechamada 63,1 71,3 74,0 [illegible] 92,33 90,3						
RF	precisão 99,1 98,7 98,7 98,8 98,8 98,7 98,7 rechamada 58,5 70,1 73,0 [illegible] 84,9 85,7 86,3 86,9 91,8 96,53 94,0 94,9						
MLP	precisão 91,3 89,1 90,7 92,2 91,6 90,0 90,5 rechamada 54,3 64,5 67,0 [illegible]						
AB	precisão 99,1 98,7 98,7 98,8 98,8 98,7 98,7 rechamada 58,5 70,1 73,0 [illegible] 94,9						

chamadas são classificadas como inofensivas. As chamadas classificadas como de baixa ameaça pelo artigo podem ser encontradas na Tabela II, que representa parte da estrutura definida por [31].

TABELA II: Chamadas de sistema classificadas como inofensivas. Obtido de [31].

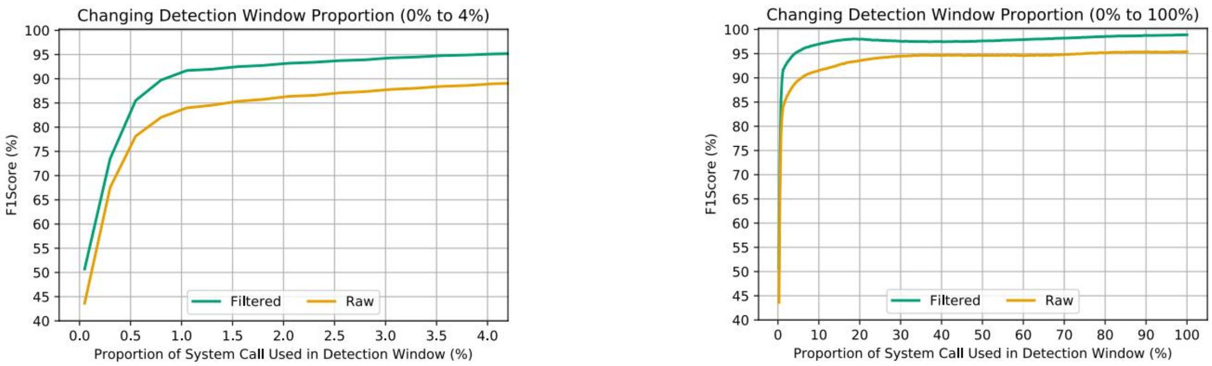
4	eu oldstat, oldfstat, acesso, sincronização, canal, ustat, oldstat, link de leitura, readdir, statfs, fstatfs, stat, getpmsg, lstat, fstat, oldname, bdflush, sysfs, getdents, fdatasync
	II getpid, getppid, getuid, getgid, geteuid, getegid, acct, getpgrp, sgetmask, getrlimit, getrusage, getgroups, getpriority, sched getscheduler, sched getparam, sched get priority min, sched rr get interval, capget, getpid, getsid, getresuid getresgid ,
	III obter kernel syms, criar módulo, módulo de consulta
	IV vezes, hora, gettimeofday, gettimer
	V sysinfo, uname
	VI ocioso
	VII break, ftime, mpx, stty, prof, ulimit, gtty, lock, profil

Esse grupo “inofensivo” consiste em chamadas de sistema que retornam algum valor ou atributo em um arquivo específico, como é o caso da syscall stat e suas variações, ou sobre recursos do sistema como getpid, getuid, gettimeofday, por exemplo. Este grupo de chamadas não é utilizado para manipulação de arquivos, memória, execução de comandos ou programas e não efetua alterações no sistema, podendo ser classificado como inofensivo em relação à segurança do sistema.

Com base nas chamadas do sistema apresentadas na Tabela II, elas são filtradas do fluxo de chamadas do sistema e não são levadas em consideração. Apesar de descartar chamadas de sistema de apenas um dos 4 níveis de ameaça, o volume de dados analisados foi dividido por dois em relação ao experimento anterior.

Isso significa que o treinamento e a decisão dos classificadores serão mais rápidos, ou seja, um ataque será detectado mais rapidamente.

A Tabela III apresenta os resultados utilizando as chamadas de sistema filtradas.



um cenário online. Os resultados usando menos chamadas do sistema (representando um cenário online/real) apresentam resultados piores do que usando os rastreamentos completos (cenário offline).

b Cenário geral. Resultados alterando a proporção de chamadas do sistema de 0% para 100% (rastreamento completo) em ambos os cenários (dados filtrados e brutos).

Fig. 2: Comparando soluções online e offline. Os resultados usando a abordagem off-line (quase o rastreamento completo) apresentam resultados muito melhores do que a abordagem on-line (com poucas chamadas de sistema).

Os resultados mostram melhorias em relação à classificação utilizando todas as chamadas do sistema apresentadas na Tabela III. Entretanto, é importante destacar que o f1-score apresentou crescimento, o que demonstra uma melhor adequação dos resultados em relação a todos os classificadores. Os classificadores que apresentaram melhor desempenho foram Random Forest e AdaBoost. Considerando que Random Forest é computacionalmente mais barato que AdaBoost, consideramos que é a melhor opção para este problema de classificação. AdaBoost e Random Forest têm resultados semelhantes, embora tenham estratégias diferentes para treinar os modelos; a única semelhança entre eles é o uso de árvores de decisão como classificadores de base.

TABELA III: Dez execuções com filtragem de chamadas (com filtro).

Métrica do Classificador	3 **Tamanho da Janela 5 7 9 11 13 15**
KNN	precisão 88,5 96,4 95,9 95,5 96,4 95,8 97,0 recall 71,4 82,0 85,3 86,9 82,6 90,8 75,1 38,9 91,5 92,3 f1-score 78,6 precisão 89,7 914,54 91,3 91,5 92,3 precisão 89,7 914,54 95
RF	precisão 99,1 99,1 99,2 99,3 99,2 99,2 99,2 rechamada 68,1 80,7 82,7 91,7 92,2 92,7 93,1 score 83,9 89,1 89,8 89,5 f1- 93,5 precisão 91,4 95,0 95
MLP	precisão 92,1 94,6 95,0 96,6 95,4 96,2 96,2 rechamada 64,1 77,5 85,4 82,9 88,8 85,7 89,4 score 75,3 81,8 89,3 83,1 f1- 89,6 precisão 89,16 91,0 44,0 9,6
AB	precisão 99,1 99,1 99,2 99,3 99,2 99,2 99,1 rechamada 68,1 80,7 82,7 91,7 92,2 92,7 93,1 score 83,9 89,1 89,8 89,5 f1- 93,5 precisão 91,4 95,0 95

C. Impacto do tamanho da

janela Para verificar o impacto do tamanho da janela deslizante em um sistema de detecção de intrusão, um novo conjunto de experimentos foi realizado. Este teste se concentra em aumentar o tamanho da janela em pequenas porções para analisar o comportamento do sistema de detecção de intrusão. As etapas de mudança de janela são relativas ao tamanho do menor rastreamento em nosso conjunto de dados, neste caso 594 (ou seja, um rastreamento com 594 chamadas de sistema). Os rastreamentos têm um tamanho médio de 8.646 chamadas de sistema, com o maior rastreamento tendo um tamanho de 27.171 chamadas de sistema.

Esses testes estão diretamente relacionados às abordagens online/offline. Uma estratégia online tende a usar janelas deslizantes para detecção de ataques em tempo real, geralmente usando janelas de observação de tamanho pequeno para detectá-los o mais rápido possível. Uma abordagem off-line tende a explorar um grande volume de dados e pode explorar todo o tamanho do rastreamento (ou seja, detectaria um ataque somente depois que ele acontecesse).

Dois experimentos foram definidos. A primeira consiste em aumentar a janela entre 0% e 4% em pequenos passos de 0,5%, representando um tamanho de crescimento de cerca de 1-2 chamadas de sistema a cada vez. Ambos os experimentos foram conduzidos utilizando apenas o classificador Random Forest, visto que foi o melhor nos experimentos anteriores. Testamos esses experimentos com as versões filtradas e brutas de nosso conjunto de dados. A Figura 2a mostra os resultados desta avaliação, demonstrando um rápido crescimento na pontuação f1 para os três primeiros valores mais baixos da janela.

Apesar da diferença entre as observações filtrando ou não as chamadas do sistema, o crescimento sempre ocorre. No entanto, a estratégia de filtrar as syscalls fornece um f1-score acima de 90% usando apenas 1% do tamanho do rastreamento (que consiste em 6 chamadas de sistema).

Finalmente, a Figura 2b mostra os resultados ao aumentar o tamanho da janela de 0% a 100% em etapas de 10%. Isso representa um crescimento de cerca de 59 a 60 chamadas de sistema a cada etapa. O cenário global mostra o impacto da filtragem de chamadas do sistema e demonstra que abordagens online podem ser realizadas, pois com apenas 10% do traço é possível obter um f1-score adequado, que não tende a variar significativamente com o crescimento da o tamanho da janela, até 100% do tamanho do traço.

VI. CONCLUSÃO

Neste artigo, a viabilidade da detecção de intrusão de anomalias em um aplicativo conteinerizado, usando a análise de sequências de chamadas do sistema. Ele avaliou o impacto da variação dos tamanhos das janelas deslizantes em diferentes métodos de detecção de anomalias;

e também avaliou uma pré-filtragem das chamadas de sistema utilizadas na detecção de anomalias, descartando chamadas de sistema consideradas inofensivas para a segurança do sistema.

Foi possível identificar uma melhora nos resultados após realizar a filtragem das chamadas do sistema avaliadas como inócuas. Embora essa melhora tenha sido pequena, mostrou-se um ponto interessante a ser estudado e avaliado. Como este experimento foi inspirado na classificação syscall feita por [31], uma nova avaliação de ameaças de chamadas de sistema também fica para trabalho futuro, uma vez que o trabalho não cobre todas as chamadas presentes em sistemas modernos. Além disso, alguns classificadores obtiveram bons resultados, mesmo quando não pré-filtraram as chamadas do sistema.

O estudo destaca a possibilidade de usar chamadas de sistema para identificar ameaças dentro de contêineres; apresentou bons resultados mesmo utilizando um conjunto menor de chamadas de sistema, permitindo a implementação de um detector de anomalias online. Assim, considerando o crescente número de aplicações que utilizam containers, nossa abordagem pode ser implementada para protegê-los contra diversos tipos de ataques.

Como trabalho futuro, novos algoritmos de extração de características podem ser considerados, para melhorar o desempenho da detecção sem aumentar o overhead do sistema, e o aumento do recall para um valor mais adequado, acima de 90%, que reduzirá o número de não relevantes eventos que estão sendo detectados. Além disso, nosso conjunto de dados também pode ser aprimorado com mais/novas amostras de comportamentos normais e de ataque, a fim de avaliar a robustez da abordagem proposta contra eles.

RECONHECIMENTO

Este estudo foi parcialmente financiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES). Os autores também agradecem ao departamento de Ciência da Computação da UFPR.

REFERÊNCIAS

[1] S. Sultan, I. Ahmad e T. Dimitriou, “Segurança de contêineres: problemas, desafios e o caminho a seguir”, IEEE Access, 2019.

[2] A. Lam, “Novo IPS para aumentar a segurança, a confiabilidade e o desempenho da rede do campus,” Boletim do Centro de Serviços de Computação, 2005.

[3] W. Yassin, NI Udzir, Z. Muda, MN Sulaiman et al., “Detecção de intrusão baseada em anomalias por meio de agrupamento de k-means e classificação de bayes naives”, em 4º Int. conf. Comput. Informática, ICOCI, 2013.

[4] B. Jain, MB Baig, D. Zhang, DE Porter e R. Sion, “Sok: Introspections on trust and the semantic gap”, no simpósio IEEE de 2014 sobre segurança e privacidade. IEEE, 2014, pp. 605–620.

[5] M. Liu, Z. Xue, X. Xu, C. Zhong e J. Chen, “Sistema de detecção de intrusão baseado em host com chamadas de sistema: revisão e tendências futuras”, ACM Computing Surveys (CSUR), 2018.

[6] RA Bridges, TR Glass-Vanderlan, MD Iannacone, MS Vincent e Q. Chen, “Uma pesquisa de sistemas de detecção de intrusão alavancando dados de host”, ACM Computing Surveys (CSUR), 2019.

[7] S. Forrest, SA Hofmeyr, A. Somayaji e TA Longstaff, “Um senso de identidade para processos unix”, em IEEE Symp. no sec. e Privacidade, 1996.

[8] S. Srinivasan, A. Kumar, M. Mahajan, D. Sitaram e S. Gupta, “Prob abilistic real-time intrusion detection system for docker containers,” in Int. Simp. no sec. em Computação e Comunicação. Primavera, 2018.

[9] C. EU. Systems, “Detecção de intrusão baseada em sequência”, http://www.cs.unm.edu/immsec/systemcalls.htm, 1998.

[10] J. Flora e N. Antunes, “Estudando a aplicabilidade da detecção de intrusão em ambientes de contêineres multilocatários”, em 2019 15ª Conferência Europeia de Computação Confiável (EDCC), 2019.

[11] AS Abed, TC Clancy e DS Levy, “Applying bag of system calls for anômalo behavior detection of applications in linux containers”, em 2015 IEEE Globecom Workshops (GC Wkshps), 2015.

[12] M. Mitchell, J. Oldham e A. Samuel, Advanced Linux Programming. Editora New Riders, 2001.

[13] J. Cespedes e P. Machata, “ltrace(1), página de manual do linux,” 2013, https://man7.org/linux/man-pages/man1/ltrace.1.html. [14] ftrace, “perf-ftrace(1) — página de manual do linux,” março de 2018, https://man7.org/linux/man-pages/man1/perf-ftrace.1.html.

[15] K. Jain e R. Sekar, "Infraestrutura em nível de usuário para interposição de chamada do sistema: uma plataforma para detecção e confinamento de intrusão." em NDSS, 2000.

[16] M. Rajagopalan, MA Hiltunen, T. Jim e RD Schlichting, “Monitoramento de chamada do sistema usando chamadas de sistema autenticadas,” IEEE Transactions on Dependable and Secure Computing, 2006.

[17] D. Merkel, “Docker: contêineres linux leves para desenvolvimento e implantação consistentes”, jornal Linux, 2014.

[18] L. Litty, Detecção de intrusão baseada em hipervisor. Universidade de Toronto, 2005.

[19] SS Durairaju, “Detecção de intrusão em ambientes conteinerizados,” 2018.

[20] P. Sharma, L. Chaufournier, P. Shenoy e Y. Tay, "Contêineres e máquinas virtuais em escala: um estudo comparativo", na 17ª Conferência Internacional de Middleware, 2016.

[21] AS Abed, C. Clancy e DS Levy, “Sistema de detecção de intrusão para aplicativos usando contêineres linux”, no Workshop Internacional sobre Segurança e Gerenciamento de Confiança. Primavera, 2015.

[22] SS Alarifi e SD Wolthusen, “Detectando anomalias em ambientes IaaS por meio da análise de chamada do sistema host da máquina virtual”, em Int. conf. para Tecnologia da Internet e Transações Seguras, 2012.

[23] Y. Liao e VR Vemuri, “Usando técnicas de categorização de texto para detecção de intrusão.” no Simpósio de Segurança USENIX, 2002.

[24] D. Yuxin, Y. Xuebing, Z. Di, D. Li e A. Zhanchao, “Representação e seleção de recursos em métodos de detecção de código malicioso com base em chamadas estáticas do sistema”, Computers & Security, 2011.

[25] W. Wang, X.-H. Guan, e X.-L. Zhang, “Modelagem de comportamentos de programas por modelos de markov ocultos para detecção de intrusão”, na Conferência Internacional de 2004 sobre Aprendizado de Máquina e Cibernética. IEEE, 2004.

[26] M. Laureano, C. Maziero e E. Jamhour, "Detecção de intrusão em ambientes de máquinas virtuais", na 30ª Conferência Euromicro, 2004. IEEE, 2004, pp. 520–525.

[27] “CVE-2014-0160”, disponível no National Vulnerability Database, CVE-ID CVE-2019-9978., maio de 2019.

[28] “CVE-2020-25213”, disponível no National Vulnerability Database, CVE-ID CVE-2020-25213., maio de 2020.

[29] “CVE-2020-12800”, disponível em Common Vulnerabilities e Ex posturas, CVE-ID CVE-2020-12800., maio de 2020.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vander plas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot e E. Duch esnay, “Scikit-learn: Machine learning in Python,” Journal of Machine Learning Research, 2011.

[31] M. Bernaschi, E. Gabrielli e LV Mancini, “Remus: um sistema operacional com segurança aprimorada”, ACM Trans. on Information and System Security (TISSEC), 2002.