Tomar um Peek: Uma Avaliação da Detecção de Anomalias Utilizando o Sistema chama por Contentores

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

***Resumo* - *O* crescimento no uso da virtualização nos últimos dez anos tem contribuído para o melhoramento desta tecnologia. A prática de implementar e gerir este tipo de ambiente isolado suscita dúvidas sobre a segurança de tais sistemas. Considerando a proximidade do hospedeiro a um contentor, as abordagens que utilizam sistemas de detecção de anomalias tentam monitorizar e detectar comportamentos inesperados. O nosso trabalho visa utilizar chamadas de sistema para identificar ameaças dentro de um ambiente de contentor, utilizando estratégias baseadas na aprendizagem de máquinas para distinguir entre comportamentos esperados e inesperados (possíveis ameaças).**

***Termos de índice* - Detecção de *intrusão*, Segurança informática, Con- tainers**

I. Introdução

A virtualização ao nível do sistema operacional, também conhecida como con- tainerização, é um tipo de virtualização em que múltiplos espaços isolados de utilizadores são fornecidos por um único núcleo. Cada espaço de utilizador, denominado *recipiente*, executa uma aplicação com as suas dependências e recursos, isolada das outras aplicações, funcionando no mesmo hospedeiro físico. Isto permite que vários recipientes com diferentes estruturas e aplicações funcionem lado a lado, partilhando o mesmo kernel do sistema operativo.

Hoje em dia, a popularidade deste tipo de virtualização levanta preocupações de segurança, devido à proximidade entre as aplicações que funcionam num contentor e o hospedeiro, porque a camada de contentorização é muito mais fina do que uma pilha de virtualização completa. Vários ataques foram encontrados e explorados nos últimos anos, tais como execução remota de código, escalada de privilégios, adulteração, entre outros [1].

A detecção de intrusão é uma forma de identificar e prevenir a actividade maliciosa num sistema. Os recursos utilizados para tais actividades 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 utilizando diferentes técnicas que podem ser baseadas na assinatura, que realiza a comparação de assinaturas com uma base conhecida de ameaças, ou baseada em anomalias, na qual o comportamento normal do sistema é previamente conhecido e os desvios do mesmo são classificados como ameaças [3].

Olhando apenas para a detecção de intrusão visando contentores, algumas diferenças poderiam ser apontadas. Os contentores são mais leves do que as máquinas virtuais e, portanto, mais fáceis e mais rápidos de gerir. Além disso, a diferença semântica entre o ambiente virtual e o anfitrião é muito pequena nos contentores, em comparação com as máquinas virtuais tradicionais [4]. Isto permite um

observador externo para recolher informações muito detalhadas sobre as aplicações que correm dentro de um contentor.

Um atacante poderia explorar diferentes técnicas para comprovar- mise, ganhar acesso, ou mesmo executar código num ambiente de contentor. Tais ataques podem ser realizados a partir de uma imagem comprometida, executando aplicações com permissões desnecessárias (por exemplo, como raiz), explorando uma vulnerabilidade na aplicação que executa o contentor, ou uma aplicação com um ambiente mal configurado [1].

Este trabalho estuda e compara a eficácia de um conjunto de métodos na identificação de actividades maliciosas num contentor. Um observador externo ao contentor recolhe informações detalhadas sobre a sua execução, sob a forma de uma sequência de chamada do sistema emitida pela aplicação. Estes dados são então utilizados para formar classificadores para construir um modelo de "comportamento normal" de cada contentor, e consequentemente permitir a identificação de anomalias.

Em resumo, este documento faz as seguintes contribuições:

- Uma discussão sobre a utilização da detecção de intrusão utilizando chamadas de sistema em contentores, e como foi implementada;
- Uma metodologia para a identificação de ataques dentro de um contentor utilizando a aprendizagem de máquinas, alcançando altas taxas de acesso e precisão para ambas as estratégias apresentadas;
- Exploramos o impacto do tamanho da janela e a filtragem da chamada do sistema para um sistema de detecção de intrusão num contentor;
- Um conjunto de dados que permite a avaliação da utilização de chamadas de sistema para detectar ameaças dentro de um contentor.

O lembrete deste documento está estruturado da seguinte forma: A Secção II apresenta o trabalho relacionado; a Secção III apresenta o terreno de apoio; a Secção IV discute a proposta do documento; a Secção V apresenta a avaliação, e a Secção VI conclui o documento.

II. Trabalho relacionado

Do ponto de vista do atacante, existem várias possibilidades - ligações de ataque a um sistema de virtualização, tais como a exploração de vir- tualização para extrair informação privada dos utilizadores, o lançamento de ataques de Negação Distribuída de Serviço (DDoS) ou a escalada da intrusão para múltiplas instâncias de VM. A literatura destaca estudos que visam monitorizar a virtualização a diferentes níveis, a fim de identificar tais acções maliciosas [5, 6].

A detecção de intrusão em ambientes de virtualização traz alguns desafios. Se o IDS for colocado dentro do hóspede, tem uma visão rica da aplicação monitorizada, mas está também exposto a atacantes que o visem. Por outro lado, se o IDS residir

978-1-6654-2744-9/21/$31.00 ©2021 IEEE

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

No entanto, ao utilizar a virtualização baseada em contentores, a lacuna semântica é grandemente reduzida, porque todos os contentores partilham o mesmo núcleo. Assim, um IDS a funcionar directamente no SO anfitrião é capaz de observar plenamente o comportamento dos processos em curso num contentor [4].

Um dos primeiros estudos que explorou o campo da detecção de intrusão baseada em chamadas de sistema foi apresentado por [7]. Propunha um método que se inspirava nos mecanismos e algo- rítmicos utilizados pelos sistemas imunitários 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. Isto inspirou o uso de sequências curtas de chamadas de sistema para definir o comportamento normal do sistema e apresentou uma forma simples e eficiente de detectar anomalias, com possíveis aplicações a cenários em tempo real.

Para abordagens baseadas na virtualização de contentores, é possível destacar [8], que apresenta um modelo para identificar anomalias em aplicações em execução no Docker. A estratégia é utilizar *n-gramas* para identificar a probabilidade de ocorrência de um evento. A experiência atinge uma precisão de até 97% para o *conjunto de dados* UNM [9]. Contudo, este conjunto de dados é muito antigo e não é representativo das aplicações actuais e dos ambientes virtuais.

Ainda no contexto dos contentores, [10] apresenta uma análise preliminar de viabilidade para a detecção de intrusão baseada em anomalias, fo- cusing nas tecnologias Docker e LXC. O artigo propõe uma análise e uma arquitectura de 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 formação em diferentes casos. O estudo treinou ambos os algoritmos com janelas de 3 a 6 *chamadas de sistema*, e calculou a inclinação da curva de crescimento, o 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 aprendizagem 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 a utilização de janelas de tamanho 3 e
4. A base de dados utilizada para validação e experimentação foi desenvolvida para o estudo, mas não está disponível ao público.

O estudo de [11], centra-se na detecção de intrusão por anomalias em ambientes de contentores, aplicando uma técnica que combina BoSC com a técnica de STIDE. A análise do comportamento do recipiente é feita após o seu encerramento, com a ajuda de uma tabela contendo todas as *chamadas* distintas *do sistema* com o respectivo número total de ocorrências. O método lê o fluxo de *chamadas de sistema* por *épocas*, e desliza uma janela de tamanho 10 através de cada *época* produzindo uma BoSC para cada janela, que é utilizada para detectar anomalias, que por sua vez é declarada se o número de disparidades na base de comportamento normal exceder um *limiar* definido. O classificador atingiu uma taxa de detecção de 100% e uma taxa de falso positivo de 0,58% para a *época* de tamanho 5.000 e *limiar* de detecção de 10% a

tamanho da *época*. A base de dados da experiência não está disponível.

III. Antecedentes

Esta secção apresenta os principais conceitos para a compreensão do trabalho, discutindo pontos tais como a detecção de anomalias, os *chamados sistemas* para IDS e a virtualização de contentores.

A. Chamadas de sistema

As chamadas de sistema são mecanismos disponíveis para a interacção entre uma aplicação (ou processo) e o núcleo do sistema operativo. Quando um programa necessita de executar operações privilegiadas, os pedidos são feitos através de *chamadas de sistema*, geralmente porque os processos ao nível do utilizador não estão autorizados a executar tais operações. Isto ocorre para uma variedade de tarefas, tais como a interacção com recursos de hardware, gestão de memória, entrada/saída pro- cessing, operações de rede, operações de ficheiros, e outras [12]. Algumas destas acções não devem estar disponíveis para todos os processos ao nível do utilizador, como o encerramento, acesso a outras áreas de memória de processos, alteração de IDs de utilizadores, e contornar políticas de controlo de acesso. O núcleo do SO implementa assim políticas de segurança que determinam quais as chamadas de sistema que podem ser chamadas por que processos ao nível do utilizador.

Devido à posição em que *as chamadas de sistema* são implementadas, a sua observação fornece informação rica sobre as actividades realizadas pelos processos ao nível do utilizador. O conjunto de chamadas de *sistema* encontradas num sistema está directamente relacionado com o Sistema Operativo (SO) e com a arquitectura utilizada. Ferramentas como *strace* e *ftrace* [13, 14], permitem mostrar a sequência de todas as chamadas de *sistema* utilizadas por um comando ou por um processo em execução.

Independentemente das abordagens utilizadas para executar código malicioso num sistema, eles exploram normalmente a interface de *chamadas do sistema* para executar operações maliciosas [15]. É apenas através desta interface que uma aplicação comprometida interage com os serviços e recursos do sistema. A monitorização de *chamadas de sistema* é uma técnica amplamente utilizada que faz uso desta característica em comum para detectar comportamentos suspeitos de uma aplicação que possa ter sido comprometida, de modo a que seja possível uma contramedida para minimizar o problema [16].

B. Contentores

Embora o conceito de contentor só tenha sido larizado popu- nos últimos anos, esta é uma técnica introduzida já nos anos 80, para realizar o "isolamento" do software utilizando a ferramenta *chroot* em sistemas Linux. Actualmente é conhecida como uma técnica de virtualização de aplicações para os mais diversos fins, geralmente associada a ferramentas populares como o docker. O contentor é um ambiente virtual que corre num único SO e permite o carregamento e execução de uma aplicação específica e as suas dependências contidas numa virtualização de um sistema operativo [17]. Desta forma, o contentor reúne os componentes necessários para a execução da aplicação, que inclui código, bibliotecas, e variáveis de ambiente, enquanto o SO anfitrião gere o acesso aos recursos de hardware como memória e processador, e todas as operações necessárias do kernel. Os contentores promovem o isolamento entre o anfitrião e os convidados, mas este isolamento não é tão forte como na plena virtualização, onde um

hypervisor é responsável pela gestão e funcionamento dos sistemas de hóspedes. O hipervisor é a camada intermédia entre as máquinas virtuais e o anfitrião, responsável pela conversão de instruções e alocação de recursos [18].

O forte isolamento proporcionado pelo hipervisor vem com um custo, em termos de processamento, utilização de memória, e facilidade de partilha de recursos. Tais restrições são muito mais leves quando se utilizam recipientes [19]. Como os contentores partilham o mesmo núcleo, uma única instância de SO é capaz de suportar múltiplos contentores isolados. Além disso, o SO virtualizado no contentor é mais leve que uma máquina virtual, tendo apenas os recursos necessários para a execução da aplicação [20].

IV. Proposta

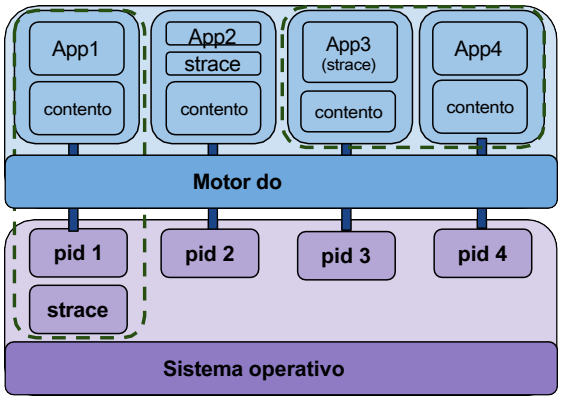
A monitorização de vulnerabilidades ao nível da aplicação é uma tarefa que consome processador, e é impraticável realizar uma monitorização a partir do interior do contentor devido às suas restrições e recursos limitados, bem como ao risco de ter o IDS exposto ao agressor [21]. Por conseguinte, uma opção seria monitorizar os processos dos hóspedes a partir do sistema anfitrião, utilizando como dados que o sistema chama, os processos gerados.

Neste contexto, [7] introduz um método de monitorização de chamadas de sistema baseado em janelas, que é simples e eficaz para a detecção em tempo real. As abordagens que utilizam tabelas de frequência [21, 22], algoritmos de Machine Learning (ML) [23, 24], e cadeias de Markov [25] também mostraram bons resultados no que diz respeito às taxas de detecção. A fim de comparar diferentes técnicas possíveis de detecção de anomalias com base em chamadas de sistema, a proposta ap- proach concentra-se numa aplicação a correr num ambiente de contentor e a ser observada a partir do hospedeiro. O nosso objectivo é explorar técnicas de ML para identificar ataques, e compreender como os sistemas de detecção de intrusão se comportarão com esta perspectiva externa. Como utilizámos uma técnica de janela deslizante, concentramo-nos em avaliar como o tamanho da janela terá impacto nos resultados, e fazemos uma avaliação minuciosa do correcto tamanho a ser utilizado.

A Figura 1 apresenta três abordagens para recolher informações de chamadas de sistema de processos de visitas dentro de um contentor. A primeira estratégia é executar o IDS ao nível do anfitrião, fora do contentor (representado por `App1` na Fig. 1). Isto protege o IDS contra um contentor comprometido [26].

A segunda abordagem consiste em utilizar a aplicação de rastreio dentro do mesmo recipiente que a aplicação está a correr (repreendida pelo `App2` na Fig. 1), o que pode ser prejudicial para o IDS. A terceira abordagem define um recipiente privilegiado para a execução do IDS (representado por `App3` e `App4` na Fig. 1). O contentor IDS deve ter direitos de acesso aos processos que estão a ser monitorizados, para reunir as chamadas ao sistema que emitem. Esta abordagem é interessante, porque permite definir um contentor IDS completo, que pode ser facilmente implantado noutro local.

Este trabalho utilizou a primeira abordagem para recolher dados, utilizando a ferramenta `strace` para recolher todas



as interacções entre o processo convidado e o kernel. Recolhemos informação suficiente para construir uma base de comportamentos normais, contendo sequências de chamadas de sistema que representam o funcionamento normal do

Fig. 1: Abordagens para monitorizar uma aplicação convidada.

e também sequências que a representam sob um ataque ou comportamento malicioso, para avaliar a proposta.

V. AVALIAÇÃO

Construímos um protótipo da nossa proposta, para verificar se a monitorização de uma aplicação hóspede do exterior do contentor é eficaz, e também para avaliar os algoritmos utilizados para detectar anomalias no fluxo de chamadas do sistema emitidas por uma aplicação. Foi também gerado um conjunto de dados rotulado de comportamento normal e ataques, que está disponível ao público[1] .

As experiências foram realizadas num anfitrião Linux utilizando o Docker como ambiente virtual. A aplicação seleccionada para os testes foi `Wordpress`, uma aplicação web que corre 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 a porta a falhas de segurança envolvendo Cross-site scripting (XSS), *SQL injection*, Remote Code Execution (RCE), entre outros.

A configuração da aplicação foi ajustada para reduzir a quantidade de fios e processos filhos bifurcados pelo servidor Apache, de modo a ter um fluxo mais limpo e mais consistente de chamadas de sistema emitidas pela aplicação. a informação de chamada de sistema recolhida contém os seus nomes, parâmetros, e valores de retorno.

O conjunto de dados construído contém cinquenta vestígios, metade do comportamento normal que consiste em interacções esperadas com a aplicação (cinco interacções diferentes), e a outra metade anomalia - comportamento lous que consiste em ataques (cinco ataques diferentes centrados em XSS e RCE). São eles: `WordPress` store XSS via `wp-admin` [27], importação/exportação CSV permitindo a execução de código PHP não autenticado, gestor de ficheiros permitindo o carregamento e execução de código PHP [28], extensões de desvio para carregar shells PHP [29], e carregamento de ficheiros a partir de utilizadores não autenticados.

[1]O conjunto de dados e o código utilizado nas experiências pode ser encontrado em https://github.com/gabrielruschel/hids-docker

Foram desenvolvidos dois grupos de testes: no primeiro caso, foram utilizadas todas as chamadas de sistema, sem aplicação de qualquer tipo de filtragem. No segundo caso, as chamadas de sistema foram classificadas em níveis de ameaça, e as menos perigosas foram descartadas.

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

A eficiência de cada algoritmo foi avaliada usando quatro métricas: precisão, recall, f1-score, e precisão. *A precisão* representa a relação entre as detecções correctamente previstas e todas as detecções que ocorreram, onde os valores elevados representam uma baixa ocorrência de falsos positivos. *A recordação*, por outro lado, representa a fracção de detecções identificadas dentro de todas as detecções possíveis. *F1-score* combina os valores de *precisão* e de *recordação* num único resultado, o que indica a qualidade global do modelo.

As secções V-A e V-B discutem as experiências realizadas e os resultados obtidos, e a secção V-C apresenta uma avaliação do tamanho da janela, considerando o impacto para a detecção de intrusão.

A. Utilização de todas as chamadas de sistema

O primeiro conjunto de experiências consiste na utilização de todas as chamadas de sistema emitidas pela aplicação. Cada chamada de sistema é representada por um identificador numérico único. A tabela I apresenta os resultados obtidos com dez execuções, sem filtragem. Em geral, os resultados parecem adequados, com precisão e precisão acima de 90% para todos os tamanhos de janelas e classificadores. A diferença entre os classificadores é bastante pequena, com uma pontuação de f1 e um número de chamada não tendo uma diferença relevante entre cada algoritmo.

Olhando apenas para o tamanho da janela na Tabela I, alguns pontos podem ser realçados: (1) os tamanhos de janela 3 e 5 apresentam os valores mais "instáveis" em relação aos outros tamanhos de janela;
(2) a diferença entre os resultados é pequena para janelas de tamanho entre 7 a 15, o que indica que poderiam apresentar um bom trade-off entre o tempo de detecção e o desempenho de classificação; e (3) *Random Forest* e *AdaBoost* apresentam os melhores resultados globais.

B. Utilização de chamadas de sistema filtradas

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

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

Classificador	Métrica	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
	recordar	63.1	71.3	74.0	76.2	76.5	77.7	79.4
	f1-score	71.4	79.7	81.3	81.8	83.4	83.5	83.3
	precisão	90.1	92.9	93.4	93.4	94.1	94.0	93.8
RF	*precisão*	99.1	98.7	98.7	98.8	98.8	98.7	98.7
	recordar	58.5	70.1	73.0	74.4	75.6	76.7	77.5
	f1-score	73.6	82.0	83.9	84.9	85.7	86.3	86.9
	precisão	91.8	94.0	94.6	94.9	95.1	95.3	95.4
MLP	*precisão*	91.3	89.1	90.7	92.2	91.6	90.0	90.5
	recordar	54.3	64.5	67.0	68.2	69.0	69.9	70.5
	f1-score	68.1	74.8	77.1	78.4	78.7	78.6	79.2
	precisão	90.1	91.5	92.2	92.7	92.7	92.6	92.8
AB	*precisão*	99.1	98.7	98.7	98.8	98.8	98.7	98.7
	recordar	58.5	70.1	73.0	74.4	75.6	76.6	77.5
	f1-score	73.6	82.0	83.9	84.9	85.7	86.3	86.8
	precisão	91.8	94.0	94.6	94.8	95.1	95.3	95.4

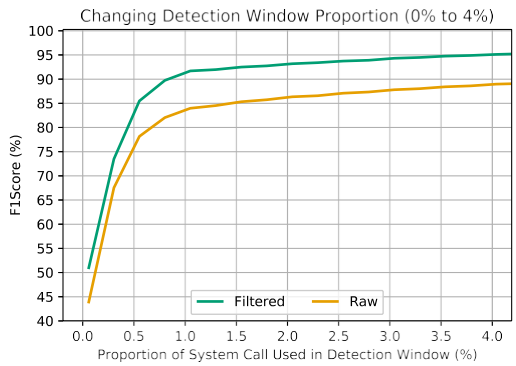
as chamadas são classificadas como inofensivas. As chamadas classificadas como de baixa ameaça pelo artigo podem ser encontradas no Quadro II, que representa parte da estrutura definida por [31].

QUADRO II: *Chamadas de sistema* classificadas como inofensivas. Obtido a partir de [31].

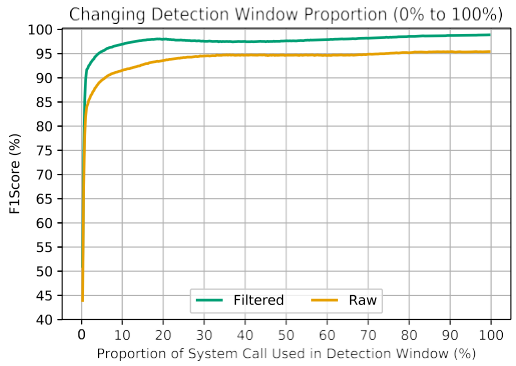
I	oldstat, oldfstat, acesso, sync, canalização, ustat, oldstat, read- link, readdir, statfs, fstatfs, stat, getpmsg, lstat, fstat, oldname, bdflush, sysfs, getdents, fdatasync
II	getpid, getppid, getuid, getgid, geteuid, getegid, acct, getpgrp, sgetmask, getrlimit, getrusage, getgroups, getpriority, sched getscheduler, sched getparam, sched getparam, sched get priority min, sched rr get interval,
4capget	, getpid, getsid, getcwdm getresgid, getresuid
III	obter os sistemas do kernel, criar módulo, módulo de consulta
IV	horas, hora, hora do dia, getitimer
V	sysinfo, uname
VI	de braços cruzados
VII	break, ftime, mpx, stty, prof, ulimit, gtty, lock, profil

Este grupo "inofensivo" consiste em chamadas de sistema que retornam algum valor ou atributo num ficheiro específico, como no caso da chamada *stat* e suas variações, ou sobre recursos do sistema como *getpid*, *getuid*, *gettimeofday*, por exemplo. Este grupo de chamadas não é utilizado para manipular ficheiros, memória, nem para executar comandos ou programas, e não faz alterações ao sistema, pelo que podem ser classificadas como inofensivas em relação à segurança do sistema.

Com base nas chamadas de sistema apresentadas no Quadro II, estas são filtradas do fluxo de chamadas do sistema e não são tidas em consideração. Apesar de descartar as chamadas de sistema de apenas um dos 4 níveis de ameaça, o volume de dados em análise foi dividido por dois em comparação com a experiência anterior. Isto significa que a formação e a decisão tomada pelos classificadores será mais rápida, ou seja, um ataque será detectado mais rapidamente. O Quadro III apresenta os resultados utilizando as chamadas de sistema filtradas.



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



b Cenário geral. Os resultados alteram a proporção de chamadas de sistema de 0% para 100% (rastreio completo) em ambos os cenários (dados filtrados e em bruto).

Fig. 2: Comparação de soluções online e offline. Os resultados utilizando a abordagem offline (quase o rastreio completo) apresentam resultados muito melhores do que a abordagem online (com poucas chamadas de sistema).

Os resultados mostram melhorias em relação à classificação utilizando todas as chamadas de sistema apresentadas no Quadro III. No entanto, é importante salientar que *a pontuação f1* apresentou um crescimento, o que demonstra uma melhor adequação dos resultados relativamente a todos os classificadores. Os classificadores que apresentaram os melhores resultados foram o *Random Forest* e o *AdaBoost*. Considerando que *Random Forest* é computacionalmente mais barato do 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 a utilização de árvores de decisão como classificadores de base.

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

Classificador	Métrica	Tamanho da janela						
		3	5	7	9	11	13	15
KNN	*precisão*	88.5	96.4	95.9	95.5	96.4	95.8	97.0
	recordar	71.4	82.0	85.1	86.1	86.8	87.5	88.0
	f1-score	78.6	88.6	90.2	90.5	91.3	91.5	92.3
	precisão	89.7	94.4	95.1	95.2	95.6	95.7	96.1
RF	*precisão*	99.1	99.1	99.2	99.3	99.2	99.2	99.2
	recordar	68.1	81.9	85.2	86.1	87.0	87.8	88.5
	f1-score	80.7	89.7	91.7	92.2	92.7	93.1	93.5
	precisão	91.4	95.0	95.9	96.2	96.4	96.6	96.8
MLP	*precisão*	92.1	94.6	95.0	96.6	95.4	96.2	96.2
	recordar	64.1	77.9	81.9	82.2	82.9	83.5	83.8
	f1-score	75.5	85.4	87.9	88.8	88.7	89.4	89.6
	precisão	89.1	93.0	94.1	94.6	94.4	94.8	94.9
AB	*precisão*	99.1	99.1	99.2	99.3	99.2	99.2	99.1
	recordar	68.1	81.9	85.2	86.1	87.0	87.8	88.5
	f1-score	80.7	89.7	91.7	92.2	92.7	93.1	93.5
	precisão	91.4	95.0	95.9	96.2	96.4	96.6	96.8

C. Impacto do tamanho da janela

Para verificar o impacto do tamanho da janela deslizante num sistema de detecção de intrusão, foi realizado um novo conjunto de experiências. Este teste centra-se no aumento do tamanho da janela em pequenas porções, a fim de analisar o comportamento do sistema de detecção de intrusão. As etapas de mudança de janela são relativas ao

tamanho do menor vestígio no nosso conjunto de dados, neste caso 594 (ou seja, um vestígio com 594 chamadas de sistema). Os vestígios têm um tamanho médio de 8, 646 chamadas de sistema, sendo que o maior vestígio tem um tamanho de 27, 171 chamadas de sistema.

Estes testes estão directamente relacionados com abordagens online/offline.

Uma estratégia online tende a utilizar janelas deslizantes para a detecção de ataques em tempo real, geralmente utilizando janelas de observação de pequeno tamanho para as detectar o mais cedo possível. Uma abordagem offline tende a explorar um grande volume de dados, e pode explorar todo o tamanho do vestígio (ou seja, só detectaria um ataque depois de este acontecer).

Foram definidas duas experiências. 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 de cada vez. Ambas as experiências foram conduzidas utilizando apenas o classificador *Random Forest*, dado que era o melhor das experiências anteriores. Testamos estas experiências com as versões filtradas e em bruto do 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 filtradas ou não, o crescimento ocorre sempre. Contudo, a estratégia de filtragem das syscalls proporciona uma *pontuação f1* acima de 90%, utilizando apenas 1% do tamanho do traço (que consiste em 6 chamadas de sistema).

Finalmente, a Figura 2b mostra os resultados quando o tamanho da janela aumenta de 0% para 100% em 10%. Este repre- senta um crescimento de cerca de 59-60 sistema chama cada passo. O cenário global mostra o impacto da filtragem de chamadas de sistema e demonstra que podem ser realizadas abordagens on-line, uma vez que utilizando apenas 10% do traço é possível obter um *f1-score* adequado, que não tende a variar significativamente com o crescimento do tamanho da janela, até 100% do tamanho do traço.

VI. Conclusão

Neste artigo, a viabilidade da detecção de intrusão de anomalias numa aplicação contentorizada, utilizando a análise de sequências de chamadas de sistema. Avaliou o impacto da variação do tamanho das janelas deslizantes em diferentes métodos de detecção de anomalias;

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

Foi possível identificar uma melhoria nos resultados após a realização da filtragem das chamadas do sistema avaliadas como inofensivas. Embora esta melhoria fosse pequena, demonstrou ser um ponto interessante a ser estudado e avaliado. Como esta experiência foi inspirada pela classificação de chamadas de sistema feita por [31], uma nova avaliação de ameaças de chamadas de sistema também permanece para trabalho futuro, uma vez que o trabalho não cobre todas as chamadas presentes nos sistemas modernos. Além disso, alguns classificadores obtiveram bons resultados, mesmo quando não pré-filtraram as chamadas do sistema.

O estudo destaca a possibilidade de utilizar chamadas de sistema para identificar ameaças dentro de contentores; apresentou bons resultados mesmo utilizando um conjunto mais pequeno de chamadas de sistema, permitindo a mentação de implementos de um detector de anomalias em linha. Assim, considerando o número crescente de aplicações que utilizam contentores, a nossa abordagem poderia ser implementada para as proteger contra muitos tipos de ataques.

Como trabalho futuro, novos algoritmos de extracção de características poderiam ser considerados, para melhorar o desempenho da detecção sem aumentar as despesas gerais do sistema, e o aumento da recolha para um valor mais adequado, superior a 90%, que reduzirá o número de eventos não relevantes a serem detectados. Além disso, o nosso conjunto de dados também poderia ser melhorado com mais/novas amostras de comportamentos normais e de ataque, a fim de avaliar a robustez da abordagem proposta contra eles.

AGRADECIMENTOS

Este estudo foi financiado em parte pela *Coordenac¸a˜o de Aperfeic¸oamento de Pessoal de N'Coordenac¸a Superior - Brasil* (CAPES). Os autores agradecem também ao departamento de Ciências da Computação da UFPR.

REFERÊNCIAS

[1] S. Sultão, I. Ahmad, e T. Dimitriou, "Segurança de contentores": Issues, challenges, and the road ahead", *IEEE Access*, 2019.

[2] A. Lam, "New IPS to boost security, reliability and performance of the campus network," *Newsletter do Centro de Serviços Informáticos*, 2005.

[3] W. Yassin, N. I. Udzir, Z. Muda, M. N. Sulaiman *et al.*, "Anomaly-based intrusion detection through k meanans clustering and naives bayes classification," in *4th Int. Conf. Comput. Informática, ICOCI*, 2013.

[4] B. Jain, M. B. Baig, D. Zhang, D. E. Porter, e R. Sion, "Sok: Introspecções sobre a confiança e a lacuna semântica", em *2014, simpósio IEEE 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 no anfitrião com chamadas de sistema": Revisão e tendências futuras", *ACM Computing Surveys (CSUR)*, 2018.

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

[7] S. Forrest, S. A. Hofmeyr, A. Somayaji, e T. A. Longstaff, "A sense of self for unix processes", in *IEEE Symp. on Sec. and Privacy*, 1996.

[8] S. Srinivasan, A. Kumar, M. Mahajan, D. Sitaram, e S. Gupta, "Probabilistic real-time intrusion detection system for docker containers," in *Int. Symp. sobre Sec. em Informática e Comunicação*. Springer, 2018.

[9] C. I. Systems, "Sequence-based intrusion detection," http://www.cs.unm.edu/ immsec/systemcalls.htm, 1998.

[10] J. Flora e N. Antunes, "Studying the applicability of intrusion detechtion to multi-tenant container environments", em *2019 15ª Conferência Europeia de Computação de Confiança (EDCC)*, 2019.

[11] A. S. Abed, T. C. Clancy, e D. S. Levy, "Applying bag of system calls para detecção de comportamento anómalo de aplicações em recipientes de linux", em *2015 IEEE Globecom Workshops (GC Wkshps)*, 2015.

[12] M. Mitchell, J. Oldham, e A. Samuel, *Advanced linux programming*. New Riders Publishing, 2001.

[13] J. Cespedes e P. Machata, "ltrace(1), linux manual page", 2013, https://man7.org/linux/man-pages/man1/ltrace.1.html.

[14] ftrace, "perf-ftrace(1) - linux manual page," mar 2018, https://man7.org/linux/man-pages/man1/perf-ftrace.1.html.

[15] K. Jain e R. Sekar, "Infra-estrutura ao nível do utilizador para interposição de chamadas de sistema: A platform for intrusion detection and confinement", em *NDSS*, 2000.

[16] M. Rajagopalan, M. A. Hiltunen, T. Jim, e R. D. Schlichting, "System call monitoring using authenticated system calls", *IEEE Transactions on Dependable and Secure Computing*, 2006.

[17] D. Merkel, "Docker: recipientes de linux leves para um desenvolvimento e implantação consistentes", *Linux journal*, 2014.

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

[19] S. S. Durairaju, "Detecção de intrusão em ambientes contentorizados", 2018.

[20] P. Sharma, L. Chaufournier, P. Shenoy, e Y. Tay, "Contentores e máquinas virtuais à escala": Um estudo comparativo", em *17th International Middleware Conference*, 2016.

[21] A. S. Abed, C. Clancy, e D. S. Levy, "Sistema de detecção de intrusão para aplicações que utilizam recipientes de linux", em *International Workshop on Security and Trust Management*. Springer, 2015.

[22] S. S. Alarifi e S. D. Wolthusen, "Detecção de anomalias em ambientes IaaS através da análise de chamadas de sistemas anfitriões de máquinas virtuais", em *Int. Conf. for Internet Technology and Secured Transactions*, 2012.

[23] Y. Liao e V. R. Vemuri, "Using text categorization techniques for intrusion detection" in *USENIX Security Symposium*, 2002.

[24] D. Yuxin, Y. Xuebing, Z. Di, D. Li, e A. Zhanchao, "Feature representation and selection in malicious code detection methods based on static system calls", *Computers & Security*, 2011.

[25] W. Wang, X.-H. Guan, e X.-L. Zhang, "Modeleling program behaviors by hidden markov models for intrusion detection", em *2004 Conferência Internacional sobre Aprendizagem de Máquinas e Cibernética*. IEEE, 2004.

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

[27] "CVE-2014-0160," Disponível na National Vulnerability Database, CVE-ID CVE-2019-9978, maio de 2019.

[28] "CVE-2020-25213," Disponível na National Vulnerability Database, CVE-ID CVE-2020-25213, maio 2020.

[29] "CVE-2020-12800," Disponível a partir de Vulnerabilidades Comuns e Posições Ex, CVE-ID CVE-2020-12800, maio 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 L. V. Mancini, "Remus: a security-enhanced operating system", *ACM Trans. on Information and System Security (TISSEC)*, 2002.