Sistema de Detecção de Intrusão para Aplicações que utilizam Containers Linux

Amr S. Abed[1] , Charles Clancy[2] , e David S. Levy[3]

[1] Departamento de Engenharia Elétrica e de Computação, Virginia Tech, Blacksburg, VA
amrabed@vt.edu
[2] Hume Center for National Security & Technology, Virginia Tech, Arlington, VA
tcc@vt.edu
[3] A Corporação MITRE, Annapolis Junction, MD
dslevy@mitre.org

Abstrato. Os contêineres Linux estão ganhando cada vez mais tração tanto no uso industrial quanto no dividual, e à medida que esses contêineres se integram aos sistemas de missão crítica, a detecção em tempo real de ataques cibernéticos maliciosos se torna um requisito operacional crítico. Este documento introduz um sistema de detecção de intrusão em tempo real baseado em host que pode ser usado para detectar passivamente prevaricação contra aplicações dentro de contêineres Linux rodando em um ambiente autônomo ou em um ambiente multi-tenancy em nuvem. O sistema de detecção de intrusão de strated demoníaco utiliza bolsas de chamadas de sistema monitoradas a partir do núcleo do host para aprender o comportamento de uma aplicação rodando dentro de um contêiner Linux e determinar o comportamento anômalo do contêiner. O desempenho da abordagem usando um aplicativo de banco de dados foi medido e os resultados são discutidos.

Palavras-chave: Detecção de Intrusão, Detecção de Anomalia, Sistema de Chamada Mon- itoring, Segurança de Contêineres, Segurança em Computação em Nuvem

1 Introdução

Os containers Linux, como Docker [13] e LXC [9], dependem dos nomes dos kernel - passos e grupos de controle (cgroups) para isolar a aplicação em execução dentro do container. Eles oferecem uma alternativa significativamente mais eficiente às máquinas virtuais, já que apenas a aplicação e suas dependências precisam ser incluídas no recipiente, e não o kernel e seus processos. Com o uso de grupos de controle e perfis de segurança aplicados aos contêineres, a superfície de ataque pode ser mini dimensionada [17]. Entretanto, ataques a aplicações de missão crítica dentro do contêiner ainda podem ocorrer, e podem representar um vetor de ataque ao próprio kernel hospedeiro [17]. Como resultado, o entendimento quando o contêiner foi comprometido é de interesse fundamental, mas pouca pesquisa tem sido conduzida nesta área.

De fato, os recipientes Linux são tipicamente usados para executar aplicações em ambientes de nuvens de múltiplos inquilinos, onde compartilham o mesmo núcleo hospedeiro com outros recipientes. Em um

ambiente multi-tenancy, o provedor de serviços tem o direito, por meios contratuais, de monitorar o comportamento dos contêineres rodando no host

kernel para fornecer um ambiente seguro para todos os recipientes hospedados e para proteger o próprio kernel do ataque de um recipiente malicioso. Entretanto, fornecer informações sobre a natureza da aplicação rodando no recipiente, ou alterar o recipiente para fins de monitoramento é geralmente indesejável, e mais freqüentemente inadmissível, especialmente quando aplicações críticas estão rodando dentro do recipiente. Tais restrições exigem o uso de um sistema de detecção de intrusão baseado em host (HIDS) que não interfira com a estrutura ou aplicação do contêiner.

Uma fonte de ataque vem de fora do hospedeiro, atacando o núcleo do hospedeiro e/ou os recipientes dos hóspedes. Outra fonte de ataque vem de outros recipientes que residem no mesmo host e que atacam os recipientes vizinhos. Uma terceira classe de ataque é quando um contêiner ataca o kernel do hospedeiro. Para atacar estes ataques, propomos um HIDS que monitora as chamadas do sistema entre os processos do contêiner e o núcleo do hospedeiro para detecção de malfeitorias.

A utilização de traços de chamada do sistema para detecção de anomalias foi previamente aplicada no nível do processo [7][11][8][14], e tem mostrado resultados promissores quando estendida à granularidade das máquinas virtuais (VMs) [15][1][2]. Ele também tem sido usado para detectar anomalias em aplicações Android através de ações de monitoramento (chamadas de sistema) incluídas em suas intenções Android [3].

Há duas abordagens básicas para a detecção de anomalias usando chamadas de sistema; abordagem baseada em seqüência e abordagem baseada em freqüência. A primeira abordagem mantém o controle das seqüências de chamadas de sistema em um banco de dados de comportamento normal. A segunda baixa a ordem das chamadas de sistema enquanto mantém a freqüência de ocorrência de cada chamada de sistema distinta. Ao não armazenar informações de ordem da seqüência de chamadas de sistema, as técnicas baseadas em freqüência requerem muito menos espaço de armazenamento, ao mesmo tempo em que proporcionam melhor desempenho e precisão [8].

Bag of system calls (BoSC) [8] é uma abordagem baseada em freqüência que tem sido usada como detectores de anomalias baseadas em VM no passado, e que foi considerado um bom executor [15][1][2]. As vantagens particulares associadas ao uso de bolsas de chamadas de sistema, em oposição às seqüências de chamadas de sistema, são que é computacionalmente manejável [1] e não requer limitar as interfaces de programação de aplicação [15].

Este documento serve para propor um HIDS em tempo real que pode ser usado para detectar passivamente anomalias de comportamento de recipientes, usando uma técnica semelhante à descrita em [1]. Mostramos que uma técnica baseada em freqüência é suficiente para desarmar anomalias no comportamento do recipiente. O sistema proposto não requer nenhum conhecimento prévio da natureza da aplicação dentro do recipiente, nem requer nenhuma alteração no recipiente nem no núcleo do hospedeiro, o que o torna o primeiro sistema a introduzir a detecção de anomalias opacas em recipientes, segundo o melhor de nosso conhecimento.

O restante deste documento está organizado da seguinte forma. A seção 2 apresenta um breve resumo do trabalho relacionado. A seção 3 fornece uma visão geral do sistema proposto. A seção 4 discute a avaliação do sistema. A seção 5 conclui com um resumo e trabalho futuro.

2 Trabalho Relacionado

A técnica *Bag of System Calls* (BoSC) é uma técnica de detecção de anomalias baseada em freqüência, que foi introduzida pela primeira vez por Kang et al. em 2005 [8]. Em seu trabalho, Kang et al. definem a bolsa de chamadas de sistema como uma lista ordenada $< c_1 , c_2 , . . c_n >$, onde n é o número total de chamadas de sistema distintas, e c_i é o número de ocorrências da chamada de sistema, s_i , na seqüência de entrada dada. Ao aplicar técnicas diferentes de aprendizagem de máquinas ferentes, tais como a classificação de 1 classe Na¨ıve Bayes e agrupamento de 2 meios, à representação do BoSC de dois conjuntos de dados de chamadas de sistema disponíveis ao público, a saber, o conjunto de dados da Universidade do Novo México (UNM) e o conjunto de dados do MIT Lincoln Lab, eles foram capazes de mostrar que o BoSC tem melhor desempenho e precisão em comparação com o STIDE [7], uma das mais famosas e mais populares abordagens baseadas em seqüência.

A técnica *Sequence Time-Delay Embedding* (STIDE), introduzida pela Forrest e Longstaff [7], define o comportamento normal usando um banco de dados de pequenas quedas, cada uma de tamanho k. Para construir o banco de dados, eles deslizam uma janela de tamanho $k + 1$ sobre o traço de chamadas de sistema, e armazenam as seqüências de chamadas de sistema. Embora STIDE seja uma técnica simples e eficiente, pode-se ver que ao manter as informações de ordem das chamadas, o tamanho do banco de dados pode crescer linearmente com o número de chamadas de sistema no rastreamento. Algumas melhorias na técnica STIDE foram introduzidas em [11] e [19].

Outra técnica famosa de detecção de intrusão baseada em seqüência é a que se utiliza em [12]. A técnica usa janelas deslizantes (regiões) de tamanho $2l + 1$, com um passo deslizante de l, e se baseia na aplicação da regra RIPPER de indução [5] para classificar seqüências de chamadas de sistema em regiões normais e anormais. Se a porcentagem de regiões anormais exceder determinado limite, o traço é declarado intrusivo.

Vários sistemas de detecção de intrusão usaram seqüências de chamadas de sistema para treinar um classificador do Modelo Markov Escondido (HMM) [19][18][20][4][10]. Entretanto, cada sistema difere na técnica usada para levantar o sinal de anomalia. Wang et al. [18], por exemplo, levantam o sinal de anomalia quando a probabilidade de toda a seqüência está abaixo de determinado limite. Warrender et al. [19], por outro lado, declaram uma seqüência como anômala quando a probabilidade de uma chamada de sistema dentro de uma se-quence está abaixo do limiar. Cho e Park [4] usaram HMM apenas para modelagem de operações normais de privilégios de raiz. Hoang et al. [10] introduziram uma técnica de detecção de multicamadas que combina os resultados da aplicação da abordagem de Janela deslizante e da abordagem HMM.

Warrender et. al compararam métodos baseados em STIDE, RIPPER e HMM em [19]. Eles concluíram que todos os métodos foram executados adequadamente, enquanto o HMM deu a melhor precisão em média. No entanto, ele exigiu maiores recursos computacionais e espaço de armazenamento, pois faz múltiplas passagens através dos dados de treinamento, e armazena uma quantidade significativa de dados intermediários, o que é computacional - aliado caro, especialmente para grandes traços.

A técnica de *Kernel State Modeling* (KSM) representa traços de chamadas de sistema como estados de módulos do Kernel [14]. A técnica observa três estados críticos, a saber: Kernel (KL), Gerenciamento de Memória (MM), e

Estados do Sistema de Arquivo (FS).

A técnica então detecta a anomalia calculando a probabilidade de ocorrências dos três estados observados em cada traço de chamada do sistema e comparando as probabilidades calculadas com as probabilidades de traços normais. Aplicada aos programas baseados em Linux do conjunto de dados UNM, a técnica KSM mostra taxas de detecção mais altas e taxas mais baixas de falsos positivos, em comparação com as técnicas baseadas em STIDE e HMM.

O sistema usado pela Alarifi e Wolthusen exige a implementação de um HIDS para máquinas virtuais residentes em um ambiente de infra-estrutura como serviço (IaaS) multi-tenancy. Eles trataram a VM como um processo único, apesar dos inúmeros processos em andamento dentro dela, e monitoraram as chamadas de sistema entre a VM e o sistema operacional do host [1] [2].

Em [1], eles usaram a técnica BoSC em combinação com a técnica de janela deslizante para a detecção de anomalias. Em sua técnica, eles lêem os traços de entrada epoch por epoch. Para cada época, uma janela deslizante de tamanho *k* se move sobre as chamadas de sistema de cada época, adicionando sacos de chamadas de sistema ao banco de dados de comportamento normal. O banco de dados de comportamento normal contém a freqüência das chamadas de sistema. Após construir o banco de dados de comportamento normal, ou seja, treinando seu classificador, uma época é declarada anômala se a mudança nas freqüências BoSC durante aquela época exceder determinado limite. Para uma janela deslizante de tamanho 10, sua técnica deu 100% de precisão, com 100% de taxa de detecção e 0% de taxa de falsos positivos.

Em [2], Alarifi e Wolthusen aplicaram HMM para seqüências de aprendizagem de chamadas de sistema para máquinas virtuais de curta duração. Eles basearam sua decisão na conclusão de [19] que "HMM quase sempre fornece uma alta taxa de detecção e um baixo mínimo de falsos positivos, mas com alta demanda computacional". Sua técnica baseada em HMM deu taxas de detecção mais baixas, mas exigiu um número menor de amostras de treinamento. Utilizando 780.000 sistemas de treinamento, a taxa de detecção resultante foi de 97%.

Em seu trabalho, Chen et al. [3] aplicaram HMM para reconhecer aplicações maliciosas do Android através de ações de monitoramento (chamadas de sistema) em intenções do Android emitidas pela aplicação. Eles concluíram que sua técnica, embora capaz de detectar aplicações Android maliciosas em tempo de execução, não tinha alto desempenho, o que eles atribuíram a não ter mensagens de intenção suficientes para treinar ainda mais o classificador.

3 Detecção de Intrusão em tempo real

Neste documento, propomos um HIDS que utiliza uma técnica semelhante à descrita em [1] para ser aplicada a recipientes Linux. A técnica combina a técnica da janela deslizante [7] com a técnica do saco de chamadas de sistema [8]. A técnica ignora a ordem das chamadas de sistema, e apenas mantém registro das freqüentes chamadas do sistema na janela atual. Como descrito na seção 1, o sistema funciona em tempo real, ou seja, aprende o comportamento do recipiente e detecta a anomalia no momento da execução. Ele também funciona em modo opaco, ou seja, não requer nenhum conhecimento prévio sobre a natureza do contêiner nem sobre a aplicação anexa. A Figura 1 dá uma visão geral da arquitetura do sistema e do fluxo de dados, conforme descrito abaixo.

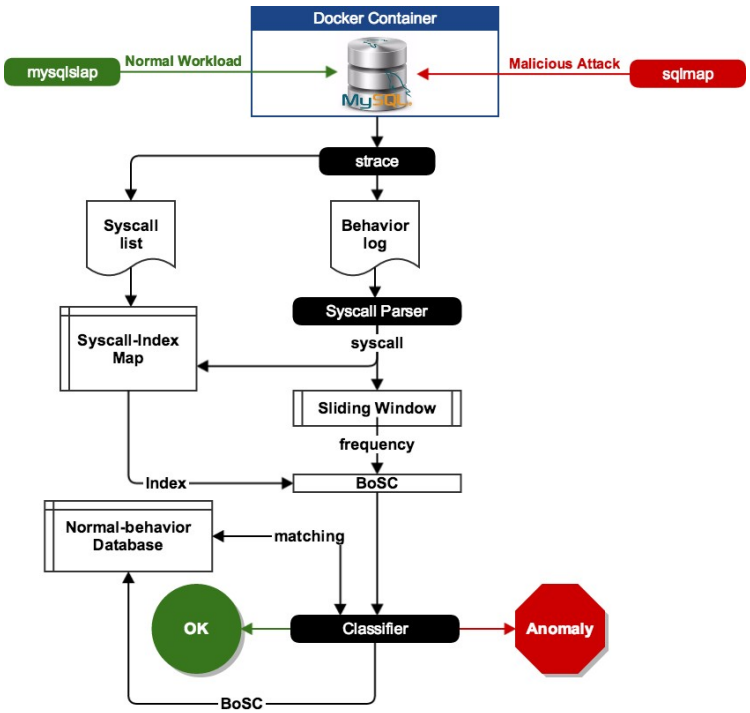

Fig. 1. Sistema de Detecção de Intrusão em Tempo Real

Nosso sistema emprega um serviço de fundo executado no kernel do host para mon- itor chamadas de sistema entre qualquer container Docker e o Kernel do host. Iniciar um novo container no kernel do host aciona o serviço, que usa a ferramenta Linux `strace` para rastrear todas as chamadas de sistema emitidas pelo container para o kernel do host. A ferramenta `strace` reporta chamadas de sistema com sua identificação do processo de origem, argu- mentos e valores de retorno.

Além disso, o `strace` também é usado para gerar um arquivo de lista de chamadas de sistema que contém uma lista pré-montada de chamadas de sistema distintas ordenadas pelo número de ocorrências. A lista é coletada de um container rodando a mesma aplicação sob nenhum ataque. O arquivo syscall-list é usado para criar uma tabela de pesquisa syscall-index. A tabela 1 mostra exemplos de entradas de uma tabela de consulta típica de índice syscall-index.

O arquivo de comportamento gerado pelo `strace` é então analisado em modo online ou offline. No modo online, o analisador de chamadas do sistema lê as chamadas do sistema a partir do mesmo arquivo que está sendo escrito pela ferramenta `strace` para classificação em tempo real. O modo off-line, por outro lado, é usado apenas para avaliação do sistema, conforme descrito na seção 4. No modo offline, uma cópia do arquivo de comportamento original é usada como entrada para o sistema para garantir a coerência entre as estatísticas coletadas. O analisador de chamadas do sistema lê uma chamada de sistema de cada vez, cortando argumentos, valores de retorno e IDs de processo.

Tabela 1. Tabela de pesquisa Syscall-Index

Syscall Index	
selecionar	4
acesso	12
lseek	22
outros	40

Tabela 2. Exemplo de análise de sistema de chamadas

Syscall	Índice	Janela deslizante	BoSC
pwrite	6	[futex, futex, sendto, futex, sendto, pwrite]	[2,0,3,0,0,0,**1**,0,...,0]
enviar para	0	[futex, sendto, futex, sendto, pwrite, sendto]	[**3**,0,**2**,0,0,0,1,0,...,0]
futex	2	[sendto, futex, sendto, pwrite, sendto, futex]	[3,0,**2**,0,0,0,1,0,...,0]
enviar para	0	[futex, sendto, pwrite, sendto, futex, sendto, sendto]	[**3**,0,2,0,0,0,1,0,...,0]

A chamada de sistema parsed é então usada para atualizar uma janela deslizante de tamanho 10 e contar o número de ocorrências de cada chamada de sistema diferente na janela atual, para criar um novo saco de chamadas de sistema. Como mencionado anteriormente, uma bolsa de chamadas de sistema é uma matriz $< c_1 , c_2 , \ldots , c_n \; s >$ onde c_i é o número de ocorrências de chamadas de sistema, s_i , na janela atual, e n_s é o número total de chamadas de sistema distintas. Quando uma nova ocorrência de uma chamada de sistema é encontrada, a aplicação recupera o índice da chamada de sistema a partir do índice syscall-index
tabela de pesquisa, e atualiza o índice correspondente do BoSC. Para uma janela tamanho 10, a soma de todas as entradas da matriz é igual a 10, ou seja $\sum_{i=1}^{n_s} c_i = 10$. A Recomenda-se geralmente o tamanho de sequência de 6 ou 10quando se usa janela deslizante
técnicas para um melhor desempenho [7][19][11]. Aqui, estamos usando 10, pois já foi mostrado para um trabalho semelhante que o tamanho 10 proporciona melhor desempenho do que o tamanho 6 sem afetar drasticamente a eficiência do algoritmo [1]. A Tabela 2 mostra um exemplo deste processo para tamanho de seqüência de 6.

O BoSC criado é então passado para o classificador, que funciona em um dos dois modos; modo de treinamento e modo de detecção. Para o modo de treinamento, o classificador simplesmente adiciona o novo BoSC ao banco de dados de comportamento normal. Se o BoSC atual já existe no banco de dados de comportamento normal, sua freqüência é incrementada por
1. Caso contrário, o novo BoSC é adicionado ao banco de dados com freqüência inicial de
1. O banco de dados de comportamento normal é considerado estável uma vez que todos os padrões de comportamento normal esperados são aplicados ao recipiente. A tabela 3 mostra exemplos de entradas de um banco de dados de comportamento normal.

Para o modo de detecção, o sistema lê o arquivo de comportamento epoch por epoch. Para cada época, uma janela deslizante é igualmente utilizada para verificar se o BoSC atual está presente no banco de dados de comportamento normal. Se um BoSC não estiver presente no banco de dados, é declarado um descasamento. O traço é declarado anômalo se o número de desajustes dentro de uma época exceder um determinado limite.

Além disso, um treinamento contínuo é aplicado durante o modo de detecção para melhorar a taxa de falsos positivos do sistema. As bolsas de chamadas do sistema

Tabela 3. Base de dados de Comportamento Normal

BoSC	Freqüência
0,1,0,2,0,0,0,0,1,0,3,0,1,0,0,0,1,0,0,1	15
0,1,0,1,0,0,1,0,1,0,3,0,0,0,0,0,1,1,0,1	8
0,1,0,2,0,0,5,0,0,0,0,0,0,0,0,0,1,0,0,1	2
0,1,0,2,0,2,0,0,1,0,2,0,0,0,0,0,1,0,0,1	1

vistas durante a época atual são armazenadas em um banco de dados temporário de mudanças de correntes, em vez de serem adicionadas diretamente ao banco de dados de comportamento normal. No final de cada época, se nenhum sinal de anomalia foi levantado durante a época atual, as entradas do banco de dados de mudança de corrente estão comprometidas com o banco de dados de comportamento normal, a ser incluído na classificação para futuras épocas.

4 Avaliação do sistema

4.1 Configuração do ambiente

Para nossos experimentos, estamos usando um container Docker rodando em um sistema operacional host Ubuntu Server 14.04. A imagem docker que usamos para criar o container é a imagem oficial do `mysql` Docker, que é basicamente uma imagem Docker com MySQL 5.6 instalado em um sistema operacional Debian.

Na partida do contêiner, o contêiner cria automaticamente um banco de dados padrão, adiciona usuários definidos pelas variáveis de ambiente passadas para o contêiner, e então começa a ouvir as conexões. O Docker mapeia a porta MySQL do contêiner para alguma porta personalizada no host.

Como não há um conjunto de dados disponível que contenha chamadas de sistema coletadas de contêineres, precisávamos criar nossos próprios conjuntos de dados tanto para o comportamento normal quanto para a anomalia. Para isso, criamos um container a partir da imagem do `mysql` Docker. Uma carga de trabalho com comportamento normal foi inicialmente aplicada ao contêiner, antes de ser "atacado" usando uma ferramenta de teste de penetração. Mais detalhes sobre a geração de conjuntos de dados são dados nas seções 4.2 e 4.3.

4.2 Geração de carga de trabalho normal

Para gerar um conjunto de dados de comportamento normal, usamos `mysqlslap` [16]; um programa que emula a carga do cliente para um servidor MySQL. A ferramenta possui opções de linha de comando que permitem ao usuário selecionar o nível de simultaneidade e o número de iterações para executar o teste de carga. Além disso, ela dá ao usuário a opção de personalizar o banco de dados criado, por exemplo, especificando o número de `varcher` e/ou colunas `int` a serem usadas ao criar o banco de dados. Além disso, o usuário pode selecionar o número de inserções e consultas a serem realizadas no banco de dados.

A ferramenta roda no kernel do host, e se comunica com o servidor MySQL rodando no container. Os valores que utilizamos para gerar a carga de trabalho de comportamento normal são mostrados na tabela 4.

Tabela 4. Parâmetros utilizados para geração automática de carga

Parâmetro	Valor
Número de colunas de `varchar` geradas	4
Número de colunas `int` geradas	3
Número de clientes simulados	50
Número de iterações de teste de carga	5
Número de declarações de inserção únicas	100
Número total de inserções por rosca	1000
Número de declarações de consulta exclusivas	100
Número total de consultas por linha	1000

Além disso, usamos o arquivo dump SQL de um banco de dados da vida real para criar esquemas, tabelas, visualizações e para adicionar entradas às tabelas, no servidor MySQL do contêiner.

4.3 Simulação de comportamento malicioso

Para simular um ataque ao recipiente, usamos o `sqlmap` [6]; uma ferramenta de injeção SQL automática normalmente usada para testes de penetração. Em nosso experimento, estamos utilizando-o para gerar dados de comportamento malicioso, atacando o banco de dados MySQL criado no contêiner. Da mesma forma, a ferramenta `sqlmap` roda no kernel do host, e se comunica com o banco de dados atacado através do proxy Docker.

Aplicamos os seguintes ataques ao contêiner:

- Ataque de Negação de Serviço (DoS): Uso de wild cards para diminuir a velocidade do banco de dados.
  O ataque gerou uma média de 37 desajustes
- Tentativa de aquisição do sistema operacional: Tentativa de executar o comando cat /etc/passwd shell (falhou). Geração de 279 desajustes
- Acesso ao sistema de arquivos: Copiar /etc/passwd para máquina local. Geração de 182 jogos errados
- Ataque por força bruta: Usamos a opção `--all` do `sqlmap` para recuperar todas as informações sobre o sistema de gerenciamento de banco de dados (SGBD), incluindo usuários, funções, esquemas, senhas, tabelas, colunas e número de entradas. O ataque foi forte o suficiente para gerar cerca de 42.000 desajustes.

4.4 Coleta de dados sobre o comportamento do recipiente

Um serviço de fundo, executado no kernel do host, detecta automaticamente qualquer container Docker recém-iniciado, e rastreia as chamadas do sistema do novo container usando a ferramenta `strace` Linux.

O serviço depende dos `eventos de` comando do Docker para sinalizar o serviço quando - sempre que um novo container é iniciado no kernel do host. Após a detecção do novo con- tainer, o serviço começa a rastrear todos os processos em execução, no início do contêiner, dentro

o grupo de controle (cgroup) do recipiente. A lista de processos é recuperada do arquivo de `tarefas` localizado em `/sys/fs/cgroup/devices/docker/$CID/tarefas`, onde `$CID` é a identificação longa do novo container. O serviço também rastreia qualquer processo de criança bifurcada usando a opção `-F` da ferramenta `strace`.

Para separar o comportamento normal do comportamento malicioso do recipiente para fins de teste, um sinal indicador é injetado no arquivo de comportamento antes e depois de cada ataque, para ser reconhecido pelo classificador.

4.5 Classificador de treinamento

Implementamos o sistema de classificação descrito na figura 1 em uma aplicação Java que utiliza a técnica descrita na seção 3.

A aplicação começa construindo um mapa hash syscall-index a partir do arquivo da lista syscall-. O mapa hash armazena chamadas de sistema distintas como a chave, e um índice correspondente como o valor. Uma chamada de sistema que aparece no traço inteiro menos do que o número total de chamadas de sistema distintas é armazenada no mapa como "outra". O uso de "outro" para chamadas de sistema relativamente raras economiza espaço, memória e tempo de computação, conforme descrito em [1]. Ao usar um mapa hash, procurar o índice de uma chamada de sistema é uma operação $O(1)$.

O analisador de chamadas do sistema lê então uma chamada de sistema por vez a partir do arquivo de registro be- havior, e atualiza o banco de dados de comportamento normal. O banco de dados de comportamento normal é outro mapa de hash com o BoSC como a chave e a freqüência da bolsa como o valor. Se a bolsa atual já existe no banco de dados, o valor da freqüência é incrementado. Caso contrário, uma nova entrada é adicionada ao banco de dados. Novamente, usando um mapa hash para implementar o banco de dados, a complexidade de tempo para atualizar o banco de dados é $O(1)$.

Como descrito na seção 3, a aplicação usa a técnica de janela deslizante para ler seqüências de chamadas do sistema a partir do arquivo de rastreamento, com cada seqüência sendo de tamanho 10. Um saco de chamadas de sistema é então criado contando a freqüência de cada chamada de sistema diferente dentro da janela atual. A bolsa de chamadas de sistema criada é uma matriz de freqüência de tamanho n_s , onde n_s é o número de chamadas de sistema distintas. Quando uma nova ocorrência de uma chamada de sistema é encontrada, a aplicação recupera o índice da chamada de sistema do mapa hash do índice syscall-index, e o índice correspondente da matriz de freqüência é atualizado. O novo BoSC é então adicionado ao banco de dados de comportamento normal.

4.6 Avaliação do classificador

O banco de dados de comportamento normal gerado é então aplicado ao resto da época do arquivo de comportamento por época para detecção de anomalias. Para cada época, a técnica de janela deslizante é utilizada de forma semelhante para criar BoSCs. Os BoSCs notados durante a época de cur- rent são adicionados ao banco de dados temporário. Um descasamento é declarado sempre que um BoSC não estiver presente no banco de dados. Se o número de desencontros exceder um determinado limite, T_d , dentro de uma época, um sinal de anomalia é aumentado. Caso contrário, as entradas do banco de dados temporário são comprometidas com o banco de dados de

comportamento normal para futuras épocas.

Para fins de avaliação, o analisador de chamadas do sistema reconhece os sinais de início e fim de ataque injetados durante a fase de coleta de dados para marcar as épocas envolvidas no ataque como maliciosas. Esta informação é usada para calcular com precisão e automaticamente a taxa positiva verdadeira (TPR) e a taxa positiva falsa (FPR) métricas, definidas como segue:

$$TPR = {}_{Ntp/Nmalicious} \quad (1)$$

$$FPR = N_{fp\,/Normal} \quad (2)$$

onde ${}_{Nnormal}$ e ${}_{Nmalicious}$ são o número total de seqüências normais e maliciosas, respectivamente, e N_{tp} e N_{fp} são o número de verdadeiros positivos e falsos positivos, respectivamente.

Para avaliar a precisão do sistema com relação aos diferentes parâmetros do sistema, aplicamos o classificador ao mesmo arquivo de comportamento de entrada, variando ao mesmo tempo os seguintes parâmetros de teste:

- Tamanho da época (S): O número total de chamadas do sistema em uma época. Para nossa experiência, utilizamos o tamanho de época entre 1000 e 10.000 com passo de 500.
- Limiar de Detecção (T_d): O número de desajustes detectados por época antes de levantar um sinal de anomalia. Usamos valores entre 10 e 100 com um degrau de 10 para cada época listada acima.

4.7 Resultados da avaliação

Aplicamos o sistema proposto a um rastreamento de 3*,* 804*,* 000 chamadas de sistema, das quais o classificador usou 875*,* 000 chamadas de sistema para treinamento. O número de chamadas de sistema distintas (n_s) era de 40, e o tamanho do banco de dados de comportamento normal era de cerca de *17k* BoSCs.

Os dados maliciosos criaram um forte sinal de anomalia com uma média de 695 desajustes por época, em comparação com uma média de 33 desajustes por época para dados normais. Para S = 1000 e T_d = *10S*, o TPR é 100% e o FPR é 2%. A Figura 2 mostra o TPR e FPR do sistema para diferentes tamanhos de época no mesmo limiar de detecção de 10. Pode-se ver que quanto menor o tamanho da época, menor é a FPR.

Como mostrado na figura 3, o limiar de detecção afeta muito a taxa de detecção do sistema, especialmente quando ataques de curta duração são introduzidos no contêiner.

4.8 Análise de complexidade

Ao utilizar o mapa hash para o mapa de índice e o banco de dados, a complexidade de tempo para procurar um índice para uma determinada chamada de sistema e para atualizar o banco de dados com um novo BoSC, são ambas operações $O(1)$. A complexidade de tempo para comparar o banco de dados antes e depois de uma época k, e para calcular a métrica de similaridade, é $O(n_k$), onde n_k é o tamanho do banco de dados após uma época k. Assim, pode ser

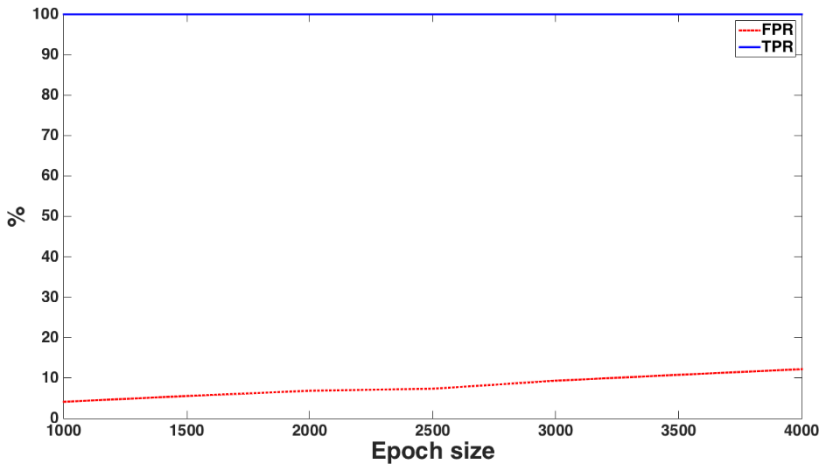

Fig. 2. Efeito da mudança do tamanho da época na precisão do sistema

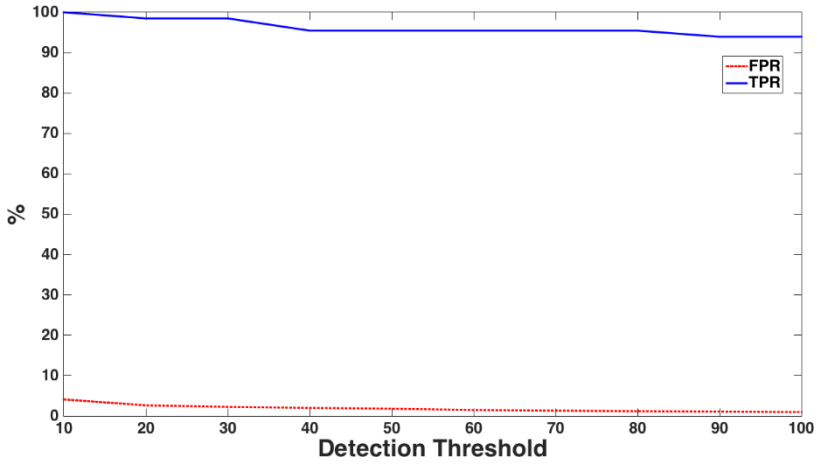

Fig. 3. Efeito da mudança do limiar de detecção na precisão do sistema

visto que o algoritmo utilizado é linear no tamanho do traço de entrada. A complexidade temporal da execução de uma época do tamanho S é $O(S + n)$.$_k$

O algoritmo usa apenas o armazenamento para o mapa de índice e para o banco de dados. O mapa de índice contém <Calça, Inteiro> pares. Assumindo que o tamanho médio do hash de chamada do sistema seja de 8 caracteres, o tamanho total de um mapa de índice de tamanho n_s é $16n_s$ bytes (Tipicamente $n_s < 50$). O banco de dados armazena um conjunto de bytes (uma string) de tamanho n_s como chave, e um número inteiro como valor. Para um banco de dados com comportamento normal de tamanho n_k , o tamanho total do banco de dados é $(n_s + 8)n_k$ bytes.

5 Conclusão e Trabalho Futuro

Neste artigo, introduzimos um sistema opaco de intrusão em tempo real baseado em host para detecção de anomalias no comportamento de recipientes Linux. O HIDS proposto utilizou uma técnica de detecção de anomalias baseada em freqüência aplicada anteriormente aos VMs. Conseguimos mostrar que uma alta taxa de detecção de 100% é facilmente alcançável utilizando um baixo limiar de detecção de 10 mismatch por época.

Embora a FPR observada fosse relativamente baixa (cerca de 2%), não fomos capazes de alcançar uma FPR zero para a aplicação usada e para a técnica de aprendizagem aplicada. Atribuímos isso ao comportamento não repetitivo da aplicação, e à natureza baseada na memória do algoritmo de aprendizagem. Foi observado que a aplicação da mesma carga de trabalho ao banco de dados MySQL pode não gerar exatamente os mesmos BoSCs, o que normalmente é esperado por uma técnica baseada em instância. O trabalho futuro deve ser direcionado para testar o sistema em aplicações de natureza repetitiva, como uma aplicação de redução de mapas, e para modificar a técnica de aprendizagem usada para ser mais adaptável a pequenas mudanças dos BoSCs gerados.

Considerando sua popularidade e simplicidade de implantação, estamos nos concentrando na fixação de contêineres Docker para esta pesquisa. Entretanto, os mesmos métodos podem ser estendidos a quaisquer outros contêineres Linux, uma vez que todos compartilham a mesma arquitetura subjacente.

Agradecimentos. Este trabalho foi financiado pela Northrop Grumman Corpora- tion através de um acordo de parceria através do S2ERC; um NSF Industry/University Cooperative Research Center. Gostaríamos de expressar nosso apreço a Don- ald Steiner e Joshua Shapiro por seu apoio e esforços de colaboração neste trabalho.

Referências

1. Alarifi, S., Wolthusen, S.: Detecção de anomalias em ambientes IaaS através da análise de chamada de sistema hospedeiro de máquina manual. In: International Conference for Internet Technology And Secured Transactions. pp. 211-218. IEEE (2012)
2. Alarifi, S., Wolthusen, S.: Detecção de anomalias para máquinas virtuais IaaS de nuvem efêmera. In: Segurança de redes e sistemas, pp. 321-335. Springer (2013)

3. Chen, Y., Ghorbanzadeh, M., Ma, K., Clancy, C., McGwier, R.: Um modelo oculto de detecção de aplicações andróides maliciosas em tempo de execução. In: 2014 23rd Wire- less and Optical Communication Conference (WOCC). pp. 1-6 (maio de 2014)
4. Cho, S.B., Park, H.J.: Detecção eficiente de anomalias através de fluxos de privilégios de modelagem usando modelo de markov oculto. Computadores e Segurança 22(1), 45 - 55 (2003)
5. Cohen, W.W.: indução rápida de regras etivas. In: Proceedings of the Twelfth Inter- national Conference on Machine Learning, Lake Tahoe, California (1995)
6. Damele, B., Stampar, M.: sqlmap: Ferramenta de injeção SQL automática e aquisição de banco de dados (2015),
7. Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T.: Um senso de si mesmo para os processos unix. In: Proceedings of the 1996 IEEE Symposium on Security and Privacy. pp. 120-128 (maio de 1996)
8. Fuller, D., Honavar, V.: Classificadores de aprendizagem para detecção de uso indevido e anomalias usando um saco de representação de chamadas do sistema. In: Anais do Sexto Workshop Anual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop. pp. 118-
125. IEEE (2005)
9. Helsley, M..: LXC: Ferramentas de contêineres Linux. Biblioteca técnica devloperWorks da IBM (2009)
10. Hoang, X.D., Hu, J., Bertok, P.: Um modelo multi-camadas para a desatenção da intrusão de anomalias utilizando seqüências de programa de chamadas de sistema. In: Proc. 11th IEEE Int?l Conf. Networks. pp. 531-536 (2003)
11. Hofmeyr, S., Forrest, S., Somayaji, A.: Detecção de intrusão usando seqüências de chamadas sys-tem. Journal of computer security 6(3), 151-180 (1998)
12. Lee, W., Stolfo, S.J.: Abordagens de mineração de dados para detecção de intrusão. In: Segurança Usenix (1998)
13. Merkel, D.: Docker: recipientes leves de linux para um desenvolvimento consistente e deployment. Linux Journal 2014(239), 2 (2014)
14. Murtaza, S.S., Khreich, W., Hamou-Lhadj, A., Couture, M.: Uma abordagem de detecção de anomalias baseada em host, representando as chamadas de sistema como estados de módulos do núcleo. In: Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Sympo- sium on. pp. 431–440. IEEE (2013)
15. Mutz, D., Valeur, F., Vigna, G., Kruegel, C.: Detecção de chamadas anômalas do sistema. ACM Transactions on Information and System Security (TISSEC) 9(1), 61-93 (2006)
16. Oracle Corporation: mysqlslap - Load Emulation Client (2015), `http://dev.mysql.com/doc/refman/5.6/pt/mysqlslap.html`
17. Petazzoni, J.: Containers & Docker: How Secure Are They? (2013), . `docker.com/2013/08/containers-docker-quale-são-seguros`
18. Wang, W., Guan, X.H., Zhang, X.L.: Modelagem de comportamentos de programa por modelos de markov escondidos para detecção de intrusão. In: Machine Learning and Cybernetics, 2004. Anais da Conferência Internacional de 2004 no vol. 5, pp. 2830-2835. IEEE
(2004)
19. Warrender, C., Forrest, S., Pearlmutter, B.: Detecção de intrusões usando chamadas de sistema: modelos de dados alternativos. In: Security and Privacy, 1999. Anais do Simpósio IEEE de 1999 em. pp. 133-145 (1999)
20. Yeung, D.Y., Ding, Y.: Detecção de intrusão baseada no hospedeiro usando modelos dinâmicos e estáticos de comportamento. Reconhecimento de padrões 36(1), 229-243 (2003)