A aplicação de saco de sistema exige detecção de comportamentos anômalos de aplicações em contêineres Linux

Amr S. Abed
Departamento de Engenharia Elétrica e de Computação Virginia Tech, Blacksburg, VA
amrabed@vt.edu

T. Charles Clancy, David S. Levy
Hume Center for National Security & Technology
Virginia Tech, Arlington, VA
{tcc, dslevy}@vt.edu

Resumo **- Neste artigo, apresentamos os resultados do uso de bolsas de sistema para aprender o comportamento de recipientes Linux para uso em sistema de detecção de intrusão baseado na detecção de anomalias. Ao utilizar chamadas de sistema dos recipientes monitorados a partir do núcleo hospedeiro para a detecção de anomalias, o sistema não requer nenhum conhecimento prévio da natureza do recipiente, nem exige a alteração do recipiente ou do núcleo hospedeiro.**

I. INTRODUÇÃO

Os containers Linux são ambientes computacionais distribuídos e gerenciados por um kernel hospedeiro. Cada contêiner normalmente executa uma única aplicação que é isolada do resto do sistema operacional. Um contêiner Linux fornece um ambiente de tempo de execução para aplicações e coleções individuais de binários e bibliotecas necessárias. Namespaces são usados para atribuir visões personalizadas, ou permissões, aplicáveis a seu ambiente de recursos necessários. Os contêineres Linux normalmente se comunicam com o núcleo do host através de chamadas ao sistema.

Ao monitorar as chamadas do sistema entre o recipiente e o núcleo hospedeiro, pode-se aprender o comportamento do recipiente a fim de detectar qualquer mudança de comportamento, o que pode refletir uma tentativa de intrusão contra o recipiente.

Uma das abordagens básicas para a detecção de anomalias usando chamadas de sistema é a técnica da *Bolsa de Chamadas de Sistema* (BoSC). A técnica BoSC é uma técnica de detecção de anomalias baseada em freqüência, que foi introduzida pela primeira vez por Kang et al. em 2005 [1]. 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. BoSC tem sido usado para
detecção de anomalias no nível do processo [1] e no nível das máquinas virtuais (VMs) [2][3][4], e tem mostrado resultados promissores.

O menor número de processos em um contêiner, em comparação com o VM, resulta em menor complexidade. A complexidade reduzida dá à técnica BoSC o potencial de ter alta precisão de detecção com um impacto marginal no desempenho do sistema quando aplicada à detecção de anomalias em contêineres.

Neste documento, estudamos a viabilidade de aplicar o BoSC para detectar passivamente ataques contra contêineres. A técnica utilizada é semelhante àquela introduzida por [3]. Mostramos

que uma técnica baseada em freqüência é suficiente para detectar anormalidades no comportamento do recipiente.

O restante deste documento está organizado da seguinte forma. A seção II fornece uma visão geral do sistema. A Seção III descreve o projeto experimental. A Seção IV discute os resultados dos experimentos. A Seção V dá um breve resumo do trabalho relacionado. A Seção VI conclui com um resumo e trabalho futuro.

II. VISÃO GERAL DO SISTEMA

Neste trabalho, usamos uma técnica semelhante à descrita em [3] aplicada aos recipientes Linux para detecção de intrusão. A técnica combina a técnica da janela deslizante [5] com a técnica da bolsa de chamadas de sistema [1], conforme descrito abaixo.

O sistema emprega um serviço de fundo executado no kernel do host para monitorar as chamadas do sistema entre quaisquer containers Docker e o Kernel do host. Ao iniciar um container, o serviço usa a ferramenta `strace` Linux para rastrear todas as chamadas de sistema emitidas pelo container para o kernel do host. O comando `strace` informa as chamadas de sistema com seu ID de processo de origem, argumentos e valores de retorno. Uma tabela de todas as chamadas de sistema distintas no rastreamento também é relatada no final do rastreamento junto com o número total de ocorrências.

O traço completo e a tabela de contagem são armazenados em um arquivo de registro que é processado off-line e usado para aprender o comportamento do contêiner após o término do contêiner. Neste ponto, não estamos realizando nenhuma aprendizagem de comportamento em tempo real ou detecção de anomalias. Portanto, lidar com todo o traço do con- tiner offline é suficiente para nossos propósitos de prova de conceito. Entretanto, para propósitos futuros, onde a aprendizagem do comportamento e a detecção de anomalias deve ser alcançada em tempo real (caso em que o traço completo não estaria disponível), o algoritmo de aprendizagem aplicado seria ligeiramente diferente do descrito aqui. Entretanto, os mesmos conceitos subjacentes continuarão a ser aplicados. O arquivo de registro gerado é então processado para criar dois arquivos, ou seja, o arquivo da lista de chamadas de sistema e o arquivo de rastreamento. O arquivo da syscall-list contém uma lista de chamadas de sistema distintas, ordenadas pelo número de ocorrências. O arquivo de rastreamento contém a lista completa de chamadas de sistema conforme coletada por argumentos, valores de retorno e IDs de processo. O arquivo de contagem é usado para criar um arquivo de

 DOI: 10.1109/CLOUDNW.2015.7114947

tabela de pesquisa syscall-index. O arquivo de rastreamento é usado para treinar o classificador, conforme descrito abaixo.

O sistema lê o arquivo de rastreamento época por época. Para cada época, uma janela deslizante de tamanho 10 é movida sobre as chamadas de sistema da época atual, contando o número de ocorrências de cada chamada de sistema distinto na janela atual, e produzindo um saco de chamadas de sistema. Como mencionado anteriormente, uma bolsa de chamadas de sistema é uma matriz $< c_1, c_2, .., c_n s >$ 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 da tabela de pesquisa do índice syscall-index, e atualiza o índice correspondente do BoSC. Para um tamanho de janela de 10, a soma de todos $\geq$ entradas da matriz igual a 10, ou seja, $\sum_{i=1}^{n} s\ c_i = 10$. Uma amostra de BoSC é mostrada abaixo para $n_s = 20$ e tamanho de sequência de 10.

```
[0,1,0,2,0,0,0,0,1,0,4,0,0,0,0,0,1,0,0,1]
```

Se o BoSC atual já existe no banco de dados de comportamento normal, sua freqüência é incrementada em 1. 1. O treinamento é concluído quando todos os padrões de comportamento normal esperados são aplicados ao sistema.

Para detectar se um traço de entrada é anômalo, o traço é lido em épocas, e para cada época, uma janela deslizante é usada 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, um descasamento é declarado. O traço é declarado anômalo se o número de desajustes dentro de uma época exceder um determinado limite.

III. PROJETO EXPERIMENTAL

A. 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 pelo ambiente vari- sáveis passados 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 comportamentos normais quanto anômalos. Para isso, criamos um container a partir da imagem do `mysql` Docker. Uma carga de trabalho de comportamento normal foi inicialmente aplicada ao contêiner, antes que ele fosse "atacado" usando uma ferramenta de teste de penetração.

B. Geração do conjunto de dados normal

Para gerar um conjunto de dados de comportamento normal, usamos `mysqlslap` [6]; 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 colunas `varcher` e/ou `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.

C. Simulação de ataque malicioso

Para simular um ataque ao recipiente, usamos o `sqlmap` [7]; 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.

D. Coleta e pré-processamento de dados

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

O serviço depende dos `eventos de` comando do Docker para sinalizar o serviço sempre que um novo container é iniciado no kernel do host. Após a detecção do novo container, o serviço começa a rastrear todos os processos em execução, no início do container, dentro do grupo de controle (cgroup) do container. O serviço também rastreia qualquer processo de criança bifurcada usando a opção `-F` da ferramenta `strace`.

O arquivo de comportamento é então passado para uma ferramenta de extração para dividir a saída em dois arquivos, ou seja, o arquivo de contagem e o arquivo de rastreamento. O arquivo de contagem contém uma lista de chamadas de sistema distintas ordenadas com base no número de ocorrências durante o rastreamento, enquanto o arquivo de rastreamento contém a lista completa de chamadas de sistema conforme coletada pelo `strace` após a remoção de argumentos, valores de retorno e informações de número de processo.

E. Modelagem do comportamento normal

Implementamos uma aplicação Java para aprender o comportamento normal do recipiente, ou seja, treinar o classificador, utilizando a técnica de detecção descrita na seção II.

A aplicação começa construindo um mapa de hash syscall-index a partir do arquivo de contagem. 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". Usar "outro" para chamadas de sistema relativamente raras economiza espaço, memória e tempo de computação, como descrito em [3].

A aplicação lê então o arquivo de vestígios em épocas. Cada época atualiza o banco de dados de comportamento normal. O banco de dados de comportamento normal é outro mapa de hash com o BoSC como 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. Para cada época, a aplicação usa a técnica de janela deslizante para ler seqüências de chamadas do sistema a partir do arquivo de rastreamento, sendo que cada seqüência é de tamanho 10. Um saco de chamadas de sistema

é então criado pela contagem da freqüência de cada chamada de sistema distinto 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.

Após executar cada época k, onde $k > 1$, o banco de dados é comparado com um instantâneo do banco de dados antes da época, e uma matriz de mudanças de freqüência BoSC, C_k , é calculada. A métrica de similaridade cosina é então usada para encontrar a similaridade entre C_k e C_{k-1} , como mostrado na equação abaixo. Dois vectores são idênticos se $\cos(\theta)$ for igual a 1.

$$\cos(\vartheta_k) = \frac{C_k \cdot C_{k-1}}{\|C^k\| \|Ck\text{-}1\|} = \frac{\sum_{i=1}^{n} C_k[i] C_{k-1}[i]}{\sqrt{\sum_{i=1}^{n} C_k[i]^2} \sqrt{\sum_{i=1}^{k-1} Ck\text{-}1[i]^2}}$$

onde $C_k[i]$ é a entrada i^{th} da matriz C_k , e n_k é o número total de entradas no banco de dados após a época k.

A condição de parada para o processo de treinamento é que a métrica de similaridade seja maior ou igual a determinado limiar, T_t , por duas épocas consecutivas.

F. Detectando anomalias

O banco de dados de comportamento normal gerado é então aplicado ao rastreamento pós-atacamento do recipiente para detecção de anomalias. Para cada época, a técnica de janela deslizante é utilizada de forma semelhante para verificar se o BoSC atual está presente no banco de dados de comportamento normal. 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.

IV. DISCUSSÃO

A. Seleção de parâmetros

Vários parâmetros foram intuitivamente selecionados ao implementar nosso sistema. Uma lista desses parâmetros é dada abaixo:

- "Outros" Threshold (T_o): O número de ocorrências de uma chamada de sistema em um rastreamento, a fim de ser marcado como "outro". Este parâmetro está atualmente definido para o número total de chamadas de sistema distintas, ou seja, o tamanho do mapa de índice de escala de sistema.
- Tamanho da época (S): O número total de chamadas do sistema em uma época. Foi observado que a relação entre o tamanho da época e o tamanho do traço de entrada afeta muito quando e se o treinamento é bem sucedido.
- Tamanho da sequência: Um tamanho sequencial de 6 ou 10 é geralmente recomendado quando se usa técnicas de janela deslizante para melhor desempenho [5][8][9].
- Limiar de Treinamento (T_t): O valor de semelhança de cosseno acima do qual declaramos dois vetores de mudança de freqüência como similares, atualmente 0,99 (selecionados por experimento).
- Limiar de Detecção (T_d): O número de erros detectados... corresponde antes de levantar um sinal de anomalia, atualmente ajustado para 10% do tamanho da época.

Investigar a correlação entre os valores dos parâmetros acima e a velocidade e precisão de detecção, e formular uma forma de selecionar seus valores ótimos, é deixado como trabalho futuro.

B. Avaliação do classificador

Para avaliar o desempenho do classificador, estamos usando as métricas de taxa positiva verdadeira (TPR) e taxa positiva falsa (FPR), definidas como segue:

$$TPR = Ntp/Nmalicious \quad (1)$$

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

Aqui, estamos usando 10, pois já foi mostrado para um trabalho semelhante que o tamanho 10 proporciona melhor desempenho que o tamanho 6 sem afetar drasticamente a eficiência do algoritmo [3].

onde N_{normal} e $N_{malicious}$ 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.

C. Resultados

Aplicamos a técnica descrita a um traço de 5, 285, 211 chamadas de sistema, usando um tamanho de época (S) de 5000. O treinador completou o treinamento após 37 épocas. O número de chamadas de sistema distintas (n_s) foi de 42, e o tamanho do banco de dados de comportamento normal foi de 10809 entradas.

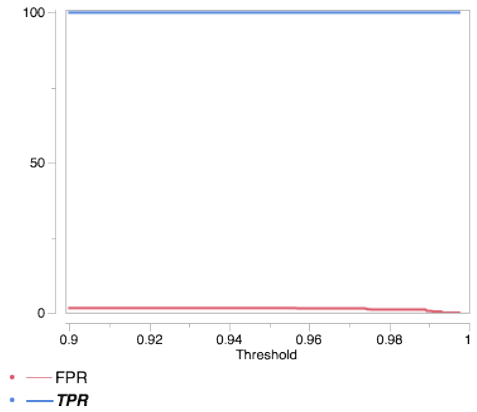
Os dados maliciosos criaram um forte sinal de anomalia com uma média de 2862 desajustes por época, enquanto os dados normais tinham uma média de 16 desajustes por época. Para $T_t = 0{,}99$ e $T_d = 0{,}15$, a taxa verdadeira positiva é de 100% e a taxa falso positiva é de 0,58%.

A Figura 1 mostra o tradeoff entre velocidade de aprendizagem e precisão. Por exemplo, a definição do limiar de treinamento (T_t) para 0,9935 baixa a FPR para 0%, às custas de aumentar o número de épocas necessárias para o treinamento para 172 épocas.

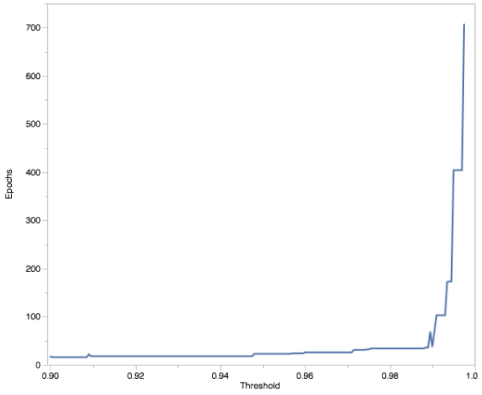
Para $T_t = 0{,}99$ e $T_d = 0{,}15$, precisávamos de um total de 185.000 chamadas de sistema para treinar o classificador. Aplicando a técnica aos VMs, o processo de treinamento precisou de 5 épocas, cada uma de 11, 743, 281 chamadas de sistema, para completar [3]. Assim, pode-se ver que a complexidade reduzida do recipiente resultou em um sistema de detecção de intrusão mais eficiente, como previsto.

V. TRABALHO RELACIONADO

O sistema usado pela Alarifi e Wolthusen exige a implementação de uma detecção de intrusão baseada em host para máquinas virtuais residentes em um ambiente de Infraestrutura como Serviço (IaaS) multi-tenancy. Eles trataram a VM como um processo único, apesar dos numerosos processos em execução dentro dela, e monitoraram as chamadas de sistema entre a VM e o sistema operacional do host [3] [4]. Em [3], eles usaram a técnica BoSC em combinação com a técnica de janela deslizante para detecção de anomalias. Em sua técnica, eles lêem o traço de entrada epoch por epoch. Para



(a) Precisão



(b) Tempo de treinamento

Fig. 1. Efeito da mudança de τ_t na precisão e no tempo de treinamento

Cada época, uma janela deslizante de tamanho *k* move-se sobre as chamadas de sistema de cada época, adicionando bolsas 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 de 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% taxa de falsos positivos.

Em [4], 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 [8] que "o HMM quase sempre fornece uma alta taxa de detecção e um 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. Ao utilizar 780.000 sistemas exige treinamento, a taxa de detecção resultante
foi de 97%.

Vários sistemas de detecção de intrusão usaram seqüências de chamadas de sistema para treinar um classificador do Modelo Markov Escondido (HMM) [8][10][11][12][13]. Entretanto, cada sistema difere na técnica usada para levantar o sinal de anomalia. Wang et al. [10], por exemplo, levantam o sinal de anomalia quando a probabilidade de toda a seqüência está abaixo de determinado limite. Warrender et

al. [8], por outro lado, declara uma seqüência como anômala quando a probabilidade de uma chamada de sistema dentro de uma seqüência está abaixo do limite. Cho e Park [12] usaram HMM apenas para modelagem de operações normais de privilégios radiculares. Hoang et al. [13] introduziram uma técnica de detecção multicamadas que combina ambos os resultados da aplicação da abordagem de Janela deslizante e da abordagem de HMM.

Warrender et. al compararam STIDE [5], RIPPER [14], e métodos baseados em HMM em [8]. Eles concluíram que todos os métodos foram executados adequadamente, enquanto que 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 computacionalmente caro, especialmente para grandes traços.

A técnica de *Kernel State Modeling* (KSM) representa traços de chamadas de sistema como estados de módulos do Kernel [15]. 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 ocorrência dos três estados observados em cada traço de chamadas de 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.

VI. CONCLUSÃO E TRABALHO FUTURO

Neste artigo, mostramos os resultados da aplicação da técnica BoSC para detecção de anomalias no comportamento de recipientes Linux. Fomos capazes de mostrar que o malicioso
criaram um forte sinal de anomalia, resultando em uma taxa de detecção de 100%, com baixa taxa de falsos positivos de 0,58%.

Utilizamos a técnica BoSC para treinar o classificador, e testar
para anomalias no modo off-line. Nosso próximo passo é modificar o algoritmo para ser mais adequado para implantação em um sistema de detecção de intrusão em tempo real. Além disso, alguns dos parâmetros atualmente utilizados pelo sistema são selecionados aleatoriamente. O trabalho futuro deve ser direcionado para estudar o efeito de tais parâmetros no processo de aprendizagem e na velocidade e precisão da detecção, numa tentativa de formalizar uma forma de otimização de seus valores para a aplicação alvo.

AGRADECIMENTOS

Este trabalho foi financiado pela Northrop Grumman Corporation através de um acordo de parceria através do S2ERC; um NSF Indus- try/University Cooperative Research Center. Gostaríamos de expressar nosso apreço a Donald Steiner e Joshua Shapiro por seu apoio e esforços de colaboração neste trabalho.

REFERÊNCIAS

[1] D. Fuller e V. Honavar, "Classificadores de aprendizagem para o uso indevido e detecção de anomalias usando um saco de representação de chamadas do sistema", em *Proceedings of the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Informa- tion Assurance Workshop*. IEEE, 2005, pp. 118-125.

[2] D. Mutz, F. Valeur, G. Vigna, e C. Kruegel, "Anomalous system call detection", *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 61-93, 2006.

[3] S. Alarifi e S. Wolthusen, "Detectando anomalias em ambientes IaaS através da análise de chamadas do sistema host de máquinas virtuais", na *Conferência Internacional para Tecnologia da Internet e Transações Seguras*. IEEE, 2012, pp. 211-218.

[4] --, "Anomaly detection for ephemeral cloud IaaS virtual machines," em *Network and system security*. Springer, 2013, pp. 321-335.

[5] S. Forrest, S. Hofmeyr, A. Somayaji, e T. Longstaff, "A sense of self for unix processes", em *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, maio de 1996, pp. 120-128.

[6] (2015) mysqlslap - Load Emulation Client. [Online]. Disponível: http://dev.mysql.com/doc/refman/5.6/en/mysqlslap.html

[7] B. Damele e M. Stampar. (2015) sqlmap: Injeção SQL automática e ferramenta de aquisição de banco de dados. [Online]. Disponível: http://sqlmap.org

[8] C. Warrender, S. Forrest, e B. Pearlmutter, "Detecting intrusions using system calls: alternative data models", em *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, 1999, pp. 133-145.

[9] S. Hofmeyr, S. Forrest, e A. Somayaji, "Detecção de intrusão usando seqüências de chamadas do sistema", *Journal of computer security*, vol. 6, no. 3,
pp. 151–180, 1998.

[10] W. Wang, X.-H. Guan, e X.-L. Zhang, "Modeling program behaviors by hidden markov models for intrusion detection", em *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, vol. 5. IEEE, 2004, pp. 2830-2835.

[11] D.-Y. Yeung e Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models", *Pattern recognition*, vol. 36, no. 1, pp. 229-243, 2003.

[12] S.-B. Cho e H.-J. Park, "Efficient anomaly detection by modeling privilege flows using hidden markov model," *Computers and Security*, vol. 22, no. 1, pp. 45 - 55, 2003.

[13] X. D. Hoang, J. Hu, e P. Bertok, "Um modelo multicamadas para detecção de intrusão de anomalias usando seqüências de programa de chamadas de sistema", no *Proc. 11th IEEE Int?l Conf. Networks*, 2003, pp. 531-536.

[14] W. Lee e S. J. Stolfo, "Data mining approaches for intrusion detection", em *Usenix Security*, 1998.

[15] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, e M. Couture, "A host-based anomaly detection approach by representing system calls as state of kernel modules," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 2013, pp. 431-440.