Detecção Eficiente de Anomalias em *Hosts* usando *Transformer* com Categorização de Chamadas de Sistema

Diogo Bortolini[1], **Rafael R. Obelheiro**[2], **Carlos A. Maziero**[1]

[1]Departamento de Informática
Universidade Federal do Paraná (UFPR) – Curitiba – PR – Brasil

[2]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC) – Joinville – SC – Brasil

{dbortolini,maziero}@inf.ufpr.br, rafael.obelheiro@udesc.br

Abstract. *This work describes an efficient approach for anomaly detection in hosts, based on the analysis of syscalls sequences. The syscalls are grouped by threat level and functionality, producing compact representations that are analyzed by a Transformer model using bigram frequency matrices. Four representations were evaluated, and the combination of threat and functionality achieved an F1-score of 0.90, close to the original model (0.92), with over 86% reduction in execution time, thus maintaining competitive performance at a lower computational cost.*

Resumo. *Este trabalho propõe uma abordagem eficiente para detecção de anomalias em hosts, baseada na análise de sequências de chamadas de sistema (syscalls). As syscalls são agrupadas por nível de ameaça e funcionalidade, gerando representações compactas analisadas por um modelo Transformer sobre matrizes de frequência de bigramas. Quatro representações foram avaliadas, e a combinação de ameaça e funcionalidade obteve F1-score de 0,90, próxima ao modelo original (0,92), com redução superior a 86% no tempo de execução, mantendo desempenho competitivo com menor custo computacional.*

1. Introdução

A segurança da informação tornou-se crucial diante do aumento da quantidade de dados, evolução tecnológica e expansão da Internet, exigindo estratégias eficazes contra ameaças cibernéticas. Neste cenário, diversas pesquisas pesquisas buscam identificar padrões anômalos nos dados do sistema para detectar intrusões de forma rápida e eficiente [Khandelwal et al. 2022].

A análise de *logs* e registros é uma abordagem tradicional, porém morosa, tediosa e complexa [Halpern 1987, Beschastnikh et al. 2020]. Isso motivou o desenvolvimento de técnicas que resumem o comportamento do sistema, facilitando sua interpretação por humanos e ferramentas automatizadas [Beschastnikh et al. 2020].

Modelos baseados em *Transformer* têm mostrado bons resultados na detecção de anomalias [Ott et al. 2021, Guan and Ezzati-Jivan 2021], mas há poucas investigações sobre métodos de redução de dimensionalidade que agrupem chamadas de sistema sem perder informações relevantes.

Este trabalho propõe uma abordagem eficiente para detectar anomalias em *hosts* usando *Transformer* e agrupamento de *syscalls* com base em risco e funcionalidade. Avalia-se o impacto desses agrupamentos na precisão e no tempo de detecção. A hipótese é que o agrupamento reduz a complexidade, priorizando comportamentos suspeitos sem perda significativa de desempenho.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta o referencial teórico; a Seção 3 discute os principais trabalhos relacionados; a Seção 4 detalha a proposta; a Seção 5 descreve a avaliação experimental; e a Seção 6 conclui o artigo.

2. Fundamentação Teórica

Para compreensão dos temas abordados neste trabalho, são apresentados os conceitos fundamentais sobre chamadas de sistemas, aprendizado de máquina e o modelo *Transformer*.

2.1. Chamadas de Sistema

As chamadas de sistema constituem a API (*Application Programming Interface*) do núcleo do sistema operacional, permitindo aos aplicativos solicitar ao núcleo a realização de tarefas específicas ou o acesso a dispositivos do hardware [Love 2010]. Cada *syscall* geralmente possui um identificador numérico único, conhecido como "número da *syscall*" e faz referência a uma chamada de sistema específica [Love 2010].

As *syscalls* fornecem uma fonte primária de informações sobre as ações dos processos no sistema, permitindo a análise detalhada de suas operações, sendo utilizadas na detecção de anomalias e violações de segurança [Vyšniūnas et al. 2024]. Desde o trabalho seminal de [Forrest et al. 1996], diversas abordagens têm explorado sua análise com esse objetivo [Liu et al. 2018, Bridges et al. 2019]. A análise de *syscalls* oferece uma granularidade mais fina que a análise de *logs*, pois esta última pode apresentar dados diluídos e irrelevantes, além da dificuldade de tratamento dos dados, já que não possuem formato, estrutura e nível de detalhe padronizados [Zhang et al. 2020].

A frequência e a ordem das *syscalls* são analisadas para classificar processos como normais ou anômalos. Entre as técnicas estão modelos de frequência relativa e n-gramas. Abordagens mais recentes aplicam conceitos de técnicas de recuperação de informação e processamento de linguagem natural (*Natural Language Processing* – NLP), tratando *syscalls* como palavras e suas sequências como documentos [Osamor and Wellman 2022].

2.2. Aprendizado de Máquina

Aprendizado de máquina permite que computadores aprendam a partir de dados, sem programação explícita, utilizando algoritmos que simulam aspectos da inteligência humana [Samuel 1959]. Redes neurais, base do aprendizado profundo (*Deep Learning* – DL), extraem representações hierárquicas por meio de arquiteturas com múltiplas camadas [Bhattacharyya et al. 2020, Bengio 2012].

O DL se destaca por identificar padrões complexos em grandes volumes de dados por meio da retropropagação [LeCun et al. 2015]. Entre suas principais arquiteturas estão as redes neurais convolucionais (*Convolutional Neural Network* – CNN), eficazes no processamento de dados em matrizes, e a redes neurais recorrentes (*Recurrent Neural Network* - RNN), projetadas para sequências temporais. Uma variação importante das

RNNs são as LSTMs (*Long Short-Term Memory*), que preservam informações por longos intervalos, sendo úteis para prever séries temporais [Deep Learning Academy 2022].

Mais recentemente, o modelo *Transformer* revolucionou o campo ao usar apenas mecanismos de atenção, eliminando convoluções e recorrências, e reduzindo o tempo de treinamento [Vaswani et al. 2017]. A autoatenção permite capturar relações contextuais complexas em sequências [Islam et al. 2024]. Desde sua introdução, o *Transformer* tem sido amplamente adotado em tarefas de NLP, visão computacional, processamento de sinais e multimodalidade [Islam et al. 2024].

3. Trabalhos Relacionados

Para contextualizar o estado da arte e identificar lacunas na literatura, foram selecionados estudos dos últimos quatro anos sobre o uso de *Transformer* na detecção de anomalias, com ênfase em abordagens baseadas em chamadas de sistema, levando em conta a relevância, originalidade e a proximidade com os objetivos deste trabalho.

[Guan and Ezzati-Jivan 2021] apresentam uma técnica híbrida do uso de LSTM e *Transformer* para a detecção de anomalias em *syscalls* em sistemas Linux, atingindo 92,6% precisão e 93,8% de revocação, superando modelos como regressão logística, SVM, florestas aleatórias e o próprio LSTM.

[Prasse et al. 2021] comparam CNN, LSTM, Florestas Aleatórias e *Transformer* na detecção de padrões de ataque em *logs* de tráfego de rede. O modelo *Transformer* superou os demais na maioria das situações, especialmente quando foi previamente treinado de forma não supervisionada.

[Fournier et al. 2023] disponibilizam um conjunto de dados com mais de dois milhões de requisições web para sete comportamentos distintos. Comparando LSTM, *Transformer* e *LongFormer* (variação do *Transformer* de menor complexidade), os modelos atingiram F1-score e AuROC maiores que 95% na maioria das classificações. *Transformer* obteve melhores resultados em três dos cinco comportamentos anômalos avaliados.

[Alshomrani et al. 2024] apresentam uma análise sobre diversos trabalhos que utilizaram *Transformer* na detecção de ameaças de segurança cibernética, destacando sua eficácia, mas também seu alto custo computacional e demanda por grandes volumes de dados.

[Ma et al. 2024] realizam uma revisão ampla sobre *Transformer* em detecção de anomalias, abordando conceitos, desafios e métricas. Identificam limitações como viés de distribuição, desequilíbrio de classes e alto consumo de recursos e velocidade lenta. Embora destaquem a possibilidade de mitigar parte desses problemas com pré-treinamento e ajustes de variantes mais eficientes com a tarefa real.

4. Proposta

Uma categorização de *syscalls* de acordo com seus níveis de ameaça e suas funcionalidades foi proposta por [Heinrich et al. 2024], com base em [Bernaschi et al. 2002]. A Tabela 1 exibe os cinco níveis de ameaça (A–E), sendo que os níveis A, B e C correspondem a chamadas com ameaça alta, enquanto D e E representam ameaça baixa. Essa classificação é baseada em correlação, não causalidade – ou seja, o uso de chamadas de alto risco não implica, por si só, comportamento malicioso, mas um processo que faz várias chamadas

no nível A, por exemplo, tende a representar ameaça maior que outro que usa apenas chamadas do nível D [Heinrich et al. 2024].

Tabela 1. Níveis de ameaças das *syscalls*. Adaptado de [Heinrich et al. 2024].

Grupo	Nível	Tipo de operação
Alta	A	Chamadas que sozinhas são mais frequentes em código malicioso que benigno.
	B	Chamadas usadas em conjunto mais frequentes em código malicioso que benigno.
	C	Chamadas que, em código malicioso, são repetidas mais vezes que em benigno.
Baixa	D	Chamadas usadas com a mesma frequência em código malicioso e em benigno, e cuja semântica permite supor que não causa violações de segurança.
	E	Chamadas não utilizadas/obsoletas.

A divisão por funcionalidade agrupa *syscalls* por áreas funcionais (dez grupos) conforme o tipo de operação: manipulação de arquivos, controle de processos, gerenciamento de módulos, gerenciamento de memória, operações de tempo, operações de comunicação, informações do sistema, manipulação de dispositivos, reservado e não implementado/removido/depuração. Tal divisão permite simplificar a análise e permitir priorizar áreas com maior risco de exploração.

A substituição das *syscalls* individuais por suas categorias reduz a dimensionalidade e o custo computacional, além de padronizar o tratamento de chamadas funcionalmente equivalentes (por exemplo, `read` e `pread`) [Das 2020]. Isso favorece a generalização dos modelos e melhora a eficiência analítica.

A proposta deste trabalho é realizar detecção de anomalias em *hosts* por meio da análise de sequências de *syscalls* usando o modelo *Transformer* [Vaswani et al. 2017]. O trabalho investiga também como o uso de representações categóricas para os dados de entrada do modelo, com base nas classificações propostas por [Heinrich et al. 2024], impacta o seu desempenho computacional (tempo para treinamento e classificação, consumo de recursos) e o seu desempenho preditivo, algo que ainda é pouco explorado na literatura.

5. Avaliação Experimental

A proposta apresentada na Seção 4 foi avaliada experimentalmente com o intuito de verificar sua aplicabilidade, sendo os resultados obtidos apresentados e discutidos.

5.1. Conjunto de dados

Para avaliação experimental, utilizou-se o conjunto de dados ADFA-LD (*Australian Defense Force Academy Linux Dataset*), gerado em um ambiente Linux realista. O conjunto contém 5.205 sequências de *syscalls* rotuladas como normais e 746 sequências como execuções anômalas, com cada *syscall* representada por um identificador inteiro entre 1 e 340 [Creech and Hu 2013].

As atividades normais envolvem desde navegação na web até configuração de serviços como Apache, MySQL, FTP, SSH e uma versão da aplicação TikiWiki com vulnerabilidades conhecidas. Já os ataques incluem força bruta (Hydra FTP/SSH), engenharia social (via Metasploit e Meterpreter), execução remota de código (falha TikiWiki), escalonamento de privilégios e inclusão remota de arquivos com webshells.

Embora ADFA-LD não seja um conjuntos de dados recente (2013), ainda é utilizado para pesquisas de detecção de anomalias, pois possui dados bem coletados e realistas, características que tornam a tarefa de aprendizado mais desafiadora e adequada para testar sistemas de detecção baseados em chamadas de sistema [Ring et al. 2021, Shin et al. 2019].

5.2. Pré-processamento

O conjunto ADFA-LD contém sequências de *syscalls*, sem atributos adicionais. Cada sequência representa a execução de um processo e está rotulada como *normal* ou *anômala*. Uma sequência S de chamadas de sistema emitidas por um processo é um vetor $S = [s_0\ s_1\ s_2\ \cdots]$, onde $s_i \in \mathbb{N}$ e $1 \leq s_i \leq max$ representa o identificador numérico de cada *syscall* (no ADFA, $max = 340$). As sequências podem ter tamanhos (número de *syscalls*) $|S|$ distintos, pois correspondem a execuções distintas. Um *n-grama* G é uma subsequência consecutiva de S com tamanho n, ou seja, um vetor $G = [s_i\ \cdots\ s_{i+n-1}]$. Mais especificamente, um *bigrama* é um n-grama com $n = 2$: $[s_i\ s_{i+1}]$.

Define-se $f(G, S)$ a frequência de ocorrências do n-grama G na sequência de *syscalls* S. No caso simplificado de bigramas, a frequência de cada bigrama em uma sequência S permite construir uma matriz de frequências $F(S)$ bidimensional com dimensões $max \times max$, onde $\forall i, j\ F_{ij}(S) = f([i\ j], S)$, ou seja, $F_{ij}(S)$ indica o número de ocorrências do bigrama $[i\ j]$ na sequência S. Essa matriz permite representar a frequência das sequências de duas chamadas de sistema consecutivas, mantendo esse atributo como parte da análise.

Este pré-processamento transforma sequências de *syscalls* variáveis em matrizes de tamanho fixo, mais adequadas para o treinamento do modelo, preservando a sequência dos dados de entrada [Hubballi 2012, Wang et al. 2017, Zhong et al. 2023]. Optou-se por bigramas devido à dimensionalidade dos dados. O uso de sequências maiores, como trigramas ou 4-gramas, implica um aumento exponencial de recursos computacionais.

5.3. Experimentos

Foram realizados experimentos com quatro variações no pré-processamento dos dados do conjunto ADFA-LD:

- *Original*: *syscalls* originais, com matrizes de frequência com dimensões 340×340;
- *Níveis*: *syscalls* substituídas pelos respectivos níveis de ameaça (Tabela 1), gerando matrizes de frequência com dimensões 5×5;
- *Funcionalidades*: *syscalls* substituídas pelas respectivas áreas funcionais, gerando matrizes de frequência com dimensões 10×10;
- *Níveis e Funcionalidades*: *syscalls* substituídas pelas combinações de níveis e funcionalidades, gerando matrizes de frequência com dimensões 50×50.

Os experimentos foram realizados em um servidor Ubuntu 24.04.01 com *kernel* 6.8.0-45, processador de 2,10 GHz, com 32 GB de RAM, 50 GB de disco, Python 3.12.3. Para cada variação, os dados foram divididos em 70% para treinamento e 30% para teste no modelo *Transformer*. Foram mensurados os tempos de pré-processamento, treinamento e teste, com repetições (mínimo de 5) e coeficiente de variação inferior a 6%.

Como o foco do estudo é o impacto do pré-processamento na representação dos dados, não foram exploradas otimizações de desempenho, outros modelos ou estratégias contra *overfitting*. Ainda assim, parâmetros do *Transformer* foram ajustados: o `d_model` foi fixado em 128, equilibrando representação e generalização, a

função `CrossEntropyLoss` foi escolhida pela eficácia em cenários desbalanceados, e foram usadas 75 épocas de treino, conforme estudos anteriores [Vaswani et al. 2017, Guan and Ezzati-Jivan 2021].

5.4. Resultados e Discussão

Foram realizados experimentos com diferentes pré-processamentos dos dados, cujos resultados estão resumidos na Tabela 2. Mediram-se tempos de execução (pré-processamento, treinamento e teste), uso máximo de memória RAM e métricas de desempenho de classificação.

Tabela 2. Resultados dos experimentos.

Categorias	Original	Níveis	Funcionalidades	Níveis e Funcionalidades
Pré-processamento (s)	11,68	1,93	2,25	2,23
Treino (s)	943,41	97,56	102,77	128,52
Teste (s)	1,60	0,14	0,13	0,12
Uso de RAM (MB)	2047	86	88	150
Precisão	0,93	0,70	0,84	0,90
Acurácia	0,97	0,83	0,94	0,96
Recall	0,91	0,85	0,92	0,91
F1 score	0,92	0,73	0,88	0,90

A representação original das *syscalls*, sem substituições, apresentou os melhores resultados (acurácia de 0,97 e F1-score de 0,92), sendo registrados apenas 25 falsos positivos e 37 falsos negativos. No entanto, com custo computacional elevado, devido à alta dimensionalidade das matrizes (340×340), resultando em 956,69 segundos de tempo total e 2.047 MB de RAM.

A substituição das *syscalls* por combinações de nível de ameaça e área funcional manteve desempenho semelhante (F1-score de 0,90), mas com redução expressiva nos recursos computacionais: tempo total de 130,87 segundos e uso de 150 MB de RAM – ganhos de mais de 7 e 13 vezes, respectivamente. Enquanto outras representações como o uso apenas da área funcional ou do nível de ameaça, apresentaram quedas mais significativas nas métricas, especialmente no último caso (F1-score de 0,73), indicando perda de informação relevante para a tarefa de detecção de anomalias.

Os resultados evidenciam o potencial do modelo *Transformer* baseado em bigramas de chamadas de sistema, destacando que o agrupamento das *syscalls* pode reduzir substancialmente os custos computacionais sem comprometer significativamente o desempenho.

6. Conclusão

Este trabalho investigou o uso do modelo *Transformer* com matrizes de frequência de bigramas de *syscalls* para detecção de anomalias. Também foram avaliadas as substituições das *syscalls* por níveis de ameaça e áreas funcionais, visando reduzir a complexidade e o volume dos dados.

A abordagem testada no conjunto ADFA-LD, apresentou menor consumo de recursos computacionais e tempos reduzidos de treinamento e teste, mantendo a eficácia na detecção de anomalias.

Como trabalhos futuros, propõe-se explorar novos agrupamentos semântico de *syscalls*, variações do modelo *Transformer* e n-gramas que considerem relações não necessariamente consecutivas entre chamadas.

Agradecimentos

Os autores agradecem o apoio da UFPR, UDESC e da FAPESC para a realização desta pesquisa.

Referências

Alshomrani, M., Albeshri, A., Alturki, B., Alallah, F. S., and Alsulami, A. A. (2024). Survey of transformer-based malicious software detection systems. *Electronics*, 13(23).

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade: 2nd ed*, pages 437–478. Springer.

Bernaschi, M., Gabrielli, E., and Mancini, L. V. (2002). REMUS: A security-enhanced operating system. *ACM Transactions on Information and System Security*, 5(1):36–61.

Beschastnikh, I., Liu, P., Xing, A., Wang, P., Brun, Y., and Ernst, M. D. (2020). Visualizing distributed system executions. *ACM Transactions on Soft. Eng. and Methodology*, 29(2).

Bhattacharyya, S., Snasel, V., Hassanien, A., Saha, S., and Tripathy, B. (2020). *Deep Learning: Research and Applications*. Frontiers in Comput. Intelligence. De Gruyter.

Bridges, R., Glass-Vanderlan, T., Iannacone, M., Vincent, ., and Chen, Q. (2019). A survey of intrusion detection systems leveraging host data. *ACM Computing Surveys*, 52(6).

Creech, G. and Hu, J. (2013). Generation of a new IDS test dataset: Time to retire the KDD collection. In *IEEE Wireless Commun. and Netw. Conf. (WCNC)*, pages 4487–4492.

Das, B. (2020). VFS over the years: An efficient change log and system call for kernel developers. *International Journal For Multidisciplinary Research*, 2(6):185–200.

Deep Learning Academy (2022). *Deep Learning Book. [s.n.].* Accessed on May 03, 2025.

Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. (1996). A sense of self for Unix processes. In *IEEE Symposium on Security and Privacy*, pages 120–128.

Fournier, Q., Aloise, D., and Costa, L. R. (2023). Language models for novelty detection in system call traces. *arXiv preprint arXiv:2309.02206*.

Guan, Y. and Ezzati-Jivan, N. (2021). Malware system calls detection using hybrid system. In *2021 IEEE International Systems Conference (SysCon)*, pages 1–8.

Halpern, J. (1987). Using reasoning about knowledge to analyze distributed systems. *Annual review of computer science*, 2(1):37–68.

Heinrich, T., Will, N. C., Obelheiro, R. R., and Maziero, C. A. (2024). A categorical data approach for anomaly detection in WebAssembly applications. In *Intl Conference on Information Systems Security and Privacy (ICISSP)*, pages 275–284.

Hubballi, N. (2012). Pairgram: Modeling frequency information of lookahead pairs for system call based anomaly detection. In *4th Intl Conference on Communication Systems and Networks (COMSNETS)*, pages 1–10.

Islam, S., Elmekki, H., Elsebai, A., Bentahar, J., Drawel, N., Rjoub, G., and Pedrycz, W. (2024). A comprehensive survey on applications of transformers for deep learning tasks. *Expert Systems with Applications*, 241:122666.

Khandelwal, P., Likhar, P., and Yadav, R. S. (2022). Machine learning methods leveraging ADFA-LD dataset for anomaly detection in Linux host systems. In *2nd Intl Conference on Intelligent Technologies (CONIT)*, pages 1–8.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444. Nature Publishing Group UK London.

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

Love, R. (2010). *Linux Kernel Development*. Developer's Library. Pearson Education.

Ma, M., Han, L., and Zhou, C. (2024). Research and application of transformer based anomaly detection model: A literature review. *arXiv preprint arXiv:2402.08975*.

Osamor, F. and Wellman, B. (2022). Deep learning-based hybrid model for efficient anomaly detection. *Intl Journal of Advanced Computer Science and Applications*.

Ott, H., Bogatinovski, J., Acker, A., Nedelkoski, S., and Kao, O. (2021). Robust and transferable anomaly detection in log data using pre-trained language models. In *IEEE/ACM Intl Workshop on Cloud Intelligence*, pages 19–24.

Prasse, P., Brabec, J., Kohout, J., Kopp, M., Bajer, L., and Scheffer, T. (2021). Learning explainable representations of malware behavior. In *European Conference on Machine Learning (ECML PKDD)*, pages 53–68. Springer.

Ring, J. H., Van Oort, C. M., Durst, S., White, V., Near, J. P., and Skalka, C. (2021). Methods for host-based intrusion detection with deep learning. *Digital Threats*, 2(4).

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

Shin, S., Lee, I., and Choi, C. (2019). Anomaly dataset augmentation using the sequence generative models. In *IEEE Intl Conf. On ML And App. (ICMLA)*, pages 1143–1148.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Vyšniūnas, T., Čeponis, D., Goranin, N., and Čenys, A. (2024). Risk-based system-call sequence grouping method for malware intrusion detection. *Electronics*, 13(1).

Wang, C., Li, Z., Mo, X., Yang, H., and Zhao, Y. (2017). An Android malware dynamic detection method based on service call co-occurrence matrices. *Annals of Telecommunications*, 72:607–615.

Zhang, X., Niyaz, Q., Jahan, F., and Sun, W. (2020). Early detection of host-based intrusions in Linux environment. In *IEEE Intl Conference EIT*, pages 475–479.

Zhong, C., Yu, Q., Luo, H., and Xie, S. (2023). A malicious programs detection method incorporating transformer and co-occurrence matrix. In *3rd Intl Conference AASIP*. International Society for Optics and Photonics, SPIE.