Classificação de Funções em Engenharia Reversa de Malware: Uma Análise Comparativa de Estratégias de Machine Learning

Vitor Reis, Elverton Fazzion, Elder Cirilo

Universidade Federal de São João del-Rei (UFSJ)

vitorejuvian@aluno.ufsj.edu.br, {fazzion,elder}@ufsj.edu.br

Abstract. *Malware analysis is a fundamental activity in cybersecurity, yet it is complex and error-prone. One of the main challenges for security analysts is distinguishing between genuinely malicious code and legitimate library code within the pseudocode extracted from executable binaries. This paper evaluates two machine learning approaches to support this classification: one based on code embeddings and another on software metrics. As a result, the syntax-based analysis via embeddings achieved 97% accuracy, outperforming the metrics-based approach, which was faster but less precise. The results demonstrate the potential of embedding analysis to support the identification of malicious code, contributing to software reverse engineering.*

Resumo. *A análise de software malicioso é uma atividade fundamental em cibersegurança, porém complexa e propensa a erros. Um dos principais desafios para os analistas de segurança é a distinção entre o código genuinamente malicioso e o código de bibliotecas legítimas presentes no pseudocódigo extraído de binários executáveis. Este artigo avalia duas estratégias de aprendizado de máquina para apoiar esta classificação: uma baseada em embeddings de código e outra em métricas de software. Como resultado, foi observado que a análise sintática via embeddings alcançou 97% de acurácia, superando a abordagem por métricas, que foi mais rápida, porém menos precisa. Os resultados demonstram o potencial da análise de embeddings para apoiar a identificação de código malicioso, contribuindo para a engenharia reversa de software.*

1. Introdução

A proliferação contínua e a crescente sofisticação de software malicioso (malware) representam uma ameaça persistente à segurança digital [Admass et al. 2024]. Nesse cenário, a engenharia reversa (ER) emerge como uma disciplina capaz de capacitar analistas de segurança a dissecar amostras de malware para compreender seus mecanismos operacionais e desenvolver contramedidas eficazes, como assinaturas de detecção e estratégias de mitigação [Kathuria et al. 2021]. Contudo, o processo de ER é intrinsecamente complexo, exigindo um alto grau de especialização técnica e um investimento de tempo considerável [Votipka et al. 2020], o que o torna um gargalo crítico nas operações de segurança cibernética. Um desafio no campo da engenharia reversa de malware surge da prática de desenvolvimento de software conhecida como vinculação estática (*static linking*). Para o analista de malware, a vinculação estática cria um severo problema de "sinal-ruído"

durante a atividade de compreensão dos mecanismos de atuação do malware. O pseudocódigo descompilado por meio de ferramentas de descompilação, como o Ghidra [1] e IDA Pro [2]; torna-se uma amálgama monolítica, onde o código malicioso está imerso em uma vasta quantidade de código de biblioteca funcionalmente complexo e que a princípio é considerado como benigno. Neste caso, como não há uma separação clara entre o código do malware e as bibliotecas, é difícil identificar quais funções pertencem ao código malicioso e quais são de bibliotecas [Li et al. 2017].

Para enfrentar o problema do ruído de código e acelerar a análise, a comunidade de pesquisa tem se concentrado no desenvolvimento de estratégias automatizadas para a triagem de funções [Shalaginov et al. 2018]. O objetivo dessas estratégias é classificar as funções descompiladas para permitir que os analistas priorizem seus esforços, focando naquelas com maior probabilidade de serem maliciosas. Neste artigo, apresentamos uma avaliação comparativa entre duas estratégias fundamentadas em aprendizado de máquina para classificação automática de funções extraídas de pseudocódigo: (i) uma baseada em análise estatística e métricas de software (e.g. complexidade ciclomática, linhas de código); e (ii) outra baseada em análise semântica via aprendizado de representação, empregando embeddings densos extraídos de tokens. O foco é analisar de modo exploratório o trade-off entre custo computacional e acurácia em cenários reais de engenharia reversa, oferecendo subsídios concretos para a adoção prática de cada abordagem em ambientes de segurança digital.

Os resultados deste estudo demonstram um claro trade-off entre custo computacional e acurácia preditiva. A análise sintática via embeddings alcançou uma acurácia de 97%, mostrando-se mais precisa que a abordagem baseada em métricas de software. Em contrapartida, a análise por métricas foi computacionalmente mais eficiente, ainda que menos precisa para a classificação de código malicioso. Este artigo está organizado da seguinte forma: a Seção 2 detalha a metodologia do estudo, descrevendo a base de dados, o modelo de classificação, as estratégias de aprendizado de máquina e as métricas de avaliação. A Seção 3 apresenta e analisa os resultados quantitativos obtidos por cada uma das abordagens investigadas. Finalmente, a Seção 4 conclui o trabalho, sumarizando os achados e indicando direções para pesquisas futuras.

2. Metodologia do Estudo Exploratório

Nesta seção, nós descrevemos o protocolo do estudo conduzido para a avaliação comparativa das estratégias de classificação. Detalhamos a construção da base de dados a partir de repositórios de malware e código benigno , o modelo de classificação selecionado e as métricas de avaliação. Por fim, apresentamos o pipeline de processamento para cada uma das abordagens investigadas.

2.1. Base de Dados

Para a realização deste estudo, nós construímos uma base de dados que contém amostras representativas de código-fonte malicioso e benigno. Como fonte para as amostras de malware, nós selecionamos o repositório IoT Malware [Ding 2021], uma coleção de códigos-fonte de malware para dispositivos de Internet das Coisas (IoT) disponibilizados

[1] https://github.com/NationalSecurityAgency/ghidra
[2] https://hex-rays.com/ida-pro

publicamente. Para a classe de código benigno, nós utilizamos o código-fonte do sistema operacional OpenBSD [The OpenBSD Project 2025], conhecido por ser um projeto de grande porte, complexo e escrito profissionalmente em linguagem C, contendo uma vasta gama de funções de biblioteca padrão e de sistema. A partir desses dois repositórios, nós extraímos os arquivos de código em linguagem C para constituir o corpus do estudo, os quais foram subsequentemente submetidos a processos de pré-processamento distintos para cada uma das estratégias investigadas.

Nós optamos por repositórios de código-fonte em vez de binários compilados, por nos permitir criar um *ground truth* de qualidade. Ao trabalhar com o código-fonte, podemos rotular cada função como maliciosa ou de biblioteca com mais precisão, eliminando a ambiguidade que surgiria na análise de binários de origem desconhecida. Embora essa abordagem abstraia os desafios de descompilação e desofuscação de binários do mundo real, ela mantém intacto o problema central de diferenciar a lógica de código malicioso da lógica de código genérico de biblioteca.

2.2. Modelo de Classificação

Nós escolhemos o *Random Forest*, um modelo reconhecido por sua robustez e desempenho em tarefas de classificação. O *Random Forest* se destaca em atividades de classificação por conta da sua capacidade de mitigar o superajuste (*overfitting*) através da construção de múltiplas árvores de decisão. Essa é uma das características desejadas para analisar os dados de alta dimensionalidade extraídos de artefatos de software e um aspecto fundamental ao lidar com a diversidade e complexidade dos padrões de código inerentes a pseudocódigos de malware e bibliotecas. Essa mesma abordagem intrínseca do *Random Forest* é adicionalmente eficaz para lidar com a intercorrelação entre métricas de software, e sua resiliência a *outliers* e valores ausentes simplifica o pré-processamento dos dados.

Para assegurar a reprodutibilidade do estudo e otimizar a precisão do modelo em face dos desafios associados à análise de pseudocódigo de malware, nós definimos e mantivemos constantes os seguintes hiperparâmetros em todos os experimentos:

- `n_estimators = 100`: Este parâmetro define o número de árvores de decisão que compõem a floresta. Nós optamos por 100 árvores para aumentar a estabilidade e a precisão da classificação por votação, um mecanismo que geralmente aprimora a performance do modelo. É crucial notar que, embora um número maior de árvores possa aprimorar a performance, ele acarreta um custo computacional mais elevado, um fator que consideramos no trade-off entre acurácia e custo na engenharia reversa de malware.
- `random_state = 42`: Para garantir a reprodutibilidade determinística do experimento, nós fixamos o estado aleatório em 42, assegurando que os resultados possam ser replicados de forma idêntica.
- `class_weight = 'balanced'`: Considerando o desbalanceamento de classes em dados, nós utilizamos a configuração 'balanced'. Este parâmetro ajusta os pesos das amostras de forma inversamente proporcional às suas frequências, forçando o modelo a dar maior importância à classe minoritária. Dada a natureza desbalanceada dos conjuntos de dados em engenharia reversa de malware, onde funções de biblioteca benignas (a classe majoritária) superam vastamente

as funções maliciosas (a classe minoritária), essa configuração é de extrema importância. Ela impede que o modelo se torne enviesado para a classe majoritária, instruindo-o a penalizar mais severamente os erros de classificação na classe minoritária (malware). Essa ponderação de classes serve como um mecanismo complementar à técnica de sobreamostragem SMOTE (*Synthetic Minority Over-sampling Technique*) [Chawla et al. 2002], que empregamos na fase de pré-processamento, para mitigar o viés do classificador e melhorar a generalização em cenários de desbalanceamento.

2.3. Estratégias de Machine Learning

Nesta seção, detalhamos as estratégias de Machine Learning que empregamos para a classificação de funções de código como maliciosas ou pertencentes a bibliotecas. Nós exploramos duas abordagens distintas e complementares: a primeira, baseada na análise semântica do código-fonte através de embeddings vetoriais; e a segunda, fundamentada em métricas de software extraídas por análise estática. Para cada estratégia, descrevemos o processo de pré-processamento dos dados, que envolveu desde a extração de funções em código C, até a coleta de um vasto conjunto de métricas quantitativas. Adicionalmente, discutimos as limitações inerentes a cada modelo, abordando o desbalanceamento de classes e os trade-offs críticos entre custo computacional, a robustez da extração de características frente a ofuscações e a dependência de ferramentas externas.

2.3.1. Classificação Semântica via Embeddings

Com essa estratégia nós investigamos a classificação de funções em nível de código-fonte por meio de embeddings semânticos. Um embedding semântico é uma representação vetorial densa, de alta dimensão, que busca encapsular o significado e a intenção funcional de um trecho de código, transcendendo sua sintaxe superficial [Jia et al. 2024]. Ao mapear cada função para um vetor em um espaço semântico, torna-se possível quantificar a similaridade funcional entre duas funções distintas, calculando a proximidade de seus respectivos vetores. Nós optamos por esta abordagem como uma contramedida às limitações das técnicas de detecção que se baseiam em características sintáticas. Tais características são geralmente frágeis diante de técnicas de ofuscação [Ebrahim and Joy 2023], que alteram a aparência do código sem modificar seu comportamento malicioso. Portanto, nossa percepção é que a representação semântica de uma função é inerentemente mais resiliente a tais manipulações, permitindo uma classificação baseada na intenção do código em vez de em padrões sintáticos facilmente manipuláveis.

Pré-processamento e Geração do Conjunto de Dados A primeira etapa do nosso pipeline de pré-processamento consiste em percorrer recursivamente as estruturas de diretórios dos repositórios (IoT Malware e OpenBSD) para identificar todos os arquivos de código-fonte escritos em linguagem C (`.c` e `.h`). Para a extração das funções individuais de cada arquivo, implementamos um *parser* que opera sobre a Árvore de Sintaxe Abstrata (AST) de cada unidade de compilação. Essa abordagem é análoga à análise semântica de um compilador e mais robusta do que a simples varredura lexical. Ao final deste processo, geramos duas coleções distintas de funções: uma contendo o código extraído dos repositórios de malware e outra com o código das bibliotecas benignas. Cada função é então

tratada como um documento individual. Para estruturar os dados para o treinamento do modelo, construímos dois *datasets*, um para cada classe, e atribuímos um rótulo binário correspondente: 1 para malware e 0 para biblioteca. Subsequentemente, estes *datasets* foram concatenados em um único conjunto de dados unificado, utilizado como entrada para a fase de geração de *embeddings* e treinamento do classificador.

2.3.2. Classificação Baseada em Métricas

Como uma abordagem ortogonal à análise semântica, nós também investigamos a eficácia da estratégia de classificação baseada em métricas de software. Esta estratégia parte da premissa de que funções maliciosas e benignas exibem características estruturais e de complexidade quantitativamente distintas. Em vez de analisar o conteúdo semântico do código, esta estratégia foca em um conjunto de indicadores numéricos – como complexidade ciclomática, métricas de volume (e.g., linhas de código) e de acoplamento – para construir um perfil para cada função [Andrade et al. 2020]. A principal justificativa para esta abordagem reside em sua eficiência computacional e interpretabilidade. Ao traduzir o código em um vetor de características numéricas, nós podemos empregar algoritmos de classificação tradicionais que são significativamente mais rápidos e menos exigentes em termos de recursos do que os modelos de linguagem de larga escala, oferecendo uma alternativa pragmática para cenários de análise em larga escala.

Pré-processamento dos dados e Geração do Conjunto de Dados Para a classificação baseada em métricas, nós utilizamos os mesmos repositórios de código-fonte (malware e bibliotecas) da estratégia anterior. A extração das características foi realizada por meio de análise estática utilizando a ferramenta comercial Understand da SciTools. Esta ferramenta analisa o código-fonte e gera um extenso conjunto de métricas de software para cada função individualmente, fornecendo dados detalhados sobre complexidade, volume e acoplamento. Com isso, o pré-processamento iniciou-se com a leitura dos arquivos CSV exportados pela ferramenta, criando um dataset para as métricas de malware e outro para as de bibliotecas. Em seguida, realizamos a limpeza dos dados, que incluiu o preenchimento de valores ausentes com zero e a conversão de tipos de dados numéricos armazenados como strings para seus formatos mais adequados (inteiro ou ponto flutuante). Após rotular cada dataset (1 para malware, 0 para biblioteca), nós os unificamos em um único conjunto de dados. Uma análise exploratória revelou um desbalanceamento de classes significativo, com uma quantidade muito maior de funções extraídas de bibliotecas do que de malwares. Para mitigar este problema, nós empregamos a técnica de sobreamostragem da classe minoritária SMOTE, que gera novas amostras sintéticas para a classe minoritária, interpolando entre instâncias vizinhas no espaço de características, em vez de simplesmente duplicar dados existentes. Dessa forma, nós fornecemos ao modelo um conjunto de treinamento mais equilibrado, o que reduz o risco de *overfitting* e melhora sua capacidade de generalização para novas amostras.

2.4. Métricas e Avaliação

Para comparar as estratégias de classificação, especialmente diante do desafio do desbalanceamento de classes inerente aos dados de malware, nós adotamos métricas

Tabela 1. Resultados - Classificação Semântica via Embeddings

	precision	recall	f1-score	support
Library	0.97	1.00	0.98	47616
Malware	0.89	0.50	0.64	2880
accuracy	0.97	0.97	0.97	0.97
macro avg	0.93	0.75	0.81	50496
weighted avg	0.97	0.97	0.96	50496

padrões. Além da acurácia geral, analisamos a *precision*, para medir a confiabilidade das classificações positivas (minimizando falsos positivos), e o *recall*, para avaliar a capacidade do modelo em identificar a totalidade das funções verdadeiramente maliciosas (minimizando ameaças não detectadas). O F1-Score foi utilizado como uma média harmônica entre *precision* e *recall*, oferecendo uma medida única e robusta para comparar o desempenho nos dados desbalanceados.

3. Resultados

Nesta seção, apresentamos os resultados do nosso estudo. O objetivo é avaliar quantitativamente a eficácia das estratégias de classificação apresentadas na Seção 2.3. A análise apresentada nesta seção foca nas métricas apresentadas na Seção 2.4 e em uma caracterização dos erros de classificação, estabelecendo assim uma linha de base para a avaliação comparativa das estratégias investigadas.

3.1. Análise de Desempenho da Classificação Semântica via Embeddings

O desempenho do modelo foi avaliado utilizando as métricas *precision*, *recall* e F1-score, cujos resultados estão consolidados na Tabela 1. Como podemos observar, para a classe

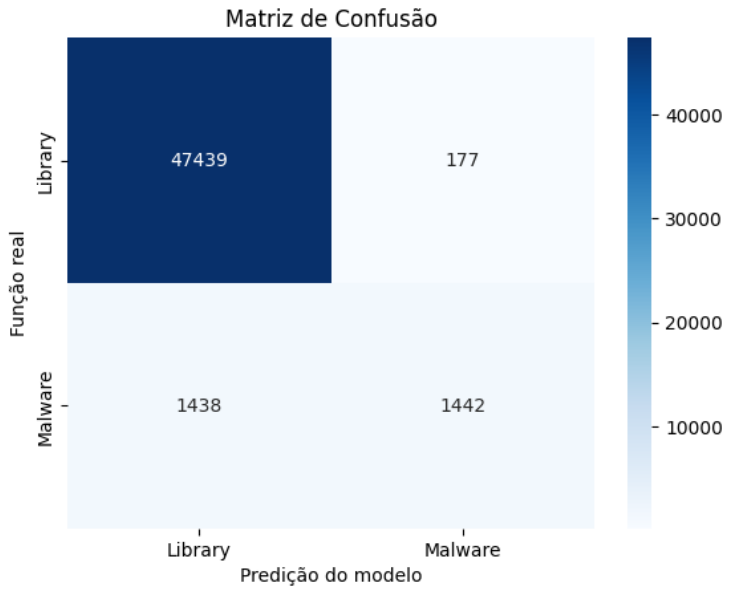

Figura 1. Matriz de Confusão - Classificação Semântica via Embeddings

de malware, o modelo alcançou uma precisão de 89%. Este valor indica que, do total de funções que o modelo previu como maliciosas, 89% estavam corretas. No entanto, a revocação para esta classe foi de apenas 50%, revelando que o modelo foi capaz de identificar corretamente apenas metade de todas as funções verdadeiramente maliciosas presentes no conjunto de teste. O F1-score resultante, de 64%, reflete um equilíbrio tênue entre precisão e revocação, o que caracteriza um desempenho mediano para a tarefa de detecção de malware. Em contrapartida, a classificação de funções de biblioteca demonstrou uma performance notavelmente superior. O modelo obteve uma precisão de 97% e uma revocação de 100%, culminando em um F1-score de 98%. Tais resultados indicam um desempenho promissor, demonstrando a capacidade do modelo em reconhecer e classificar corretamente as funções benignas.

Porém, uma análise detalhada dos erros é apresentada na matriz de confusão (Figura 1). O modelo obteve um total de 48.881 classificações corretas e 1.615 erros. A natureza desses erros é fundamental para a avaliação do risco associado ao modelo. Observamos 177 erros do tipo falso positivo, nos quais funções de biblioteca foram incorretamente classificadas como malware. As consequências deste tipo de erro são operacionais, resultando em uma perda de tempo para o analista que investigaria código benigno. No entanto, foram registrados 1.438 erros do tipo falso negativo, onde funções de malware foram erroneamente classificadas como pertencentes a bibliotecas. Este tipo de erro acarreta consequências de segurança significativamente mais graves. Funções maliciosas, que deveriam ser objeto de estudo prioritário, seriam potencialmente ignoradas pelo analista devido à classificação equivocada, representando um risco substancial de que ameaças reais não sejam detectadas.

3.2. Análise de Desempenho da Classificação Baseada em Métricas

Os resultados da estratégia de classificação baseada em métricas estão detalhados na Tabela 2. O modelo alcançou uma acurácia global de 94%. No entanto, em cenários com acentuado desbalanceamento de classes, a acurácia global pode ser uma métrica enganosa, pois é fortemente influenciada pelo desempenho do modelo na classe majoritária. Uma análise desagregada por classe revela, de fato, um desempenho marcadamente assimétrico. Para a classe de bibliotecas (majoritária), a precisão foi de 98%. Em contraste, a precisão para a classe de malware (minoritária) decaiu para 22%. Este baixo valor de precisão indica que a maioria das funções que o modelo identifica como maliciosas são, na realidade, benignas, resultando em uma alta taxa de falsos positivos. Tal comportamento é um sintoma clássico de um classificador que desenvolveu um viés em favor da classe majoritária, uma vez que algoritmos de aprendizado padrão otimizam a acurácia geral e, consequentemente, falham em aprender os padrões distintivos da classe minoritária com representação insuficiente.

Embora o classificador Random Forest seja um método de ensemble robusto, sua eficácia pode ser comprometida por distribuições de classe severamente desbalanceadas. A aplicação da técnica SMOTE, que visa mitigar esse viés pela geração de amostras sintéticas da classe minoritária , não produziu o resultado esperado. A queda acentuada na precisão sugere que as amostras sintéticas geradas pelo SMOTE podem ter introduzido ruído ou criado regiões de decisão sobrepostas, confundindo o classificador Random Forest e levando-o a generalizar de forma inadequada os limites entre as classes.

A matriz de confusão (Figura 2) fornece uma visão absoluta dos erros de

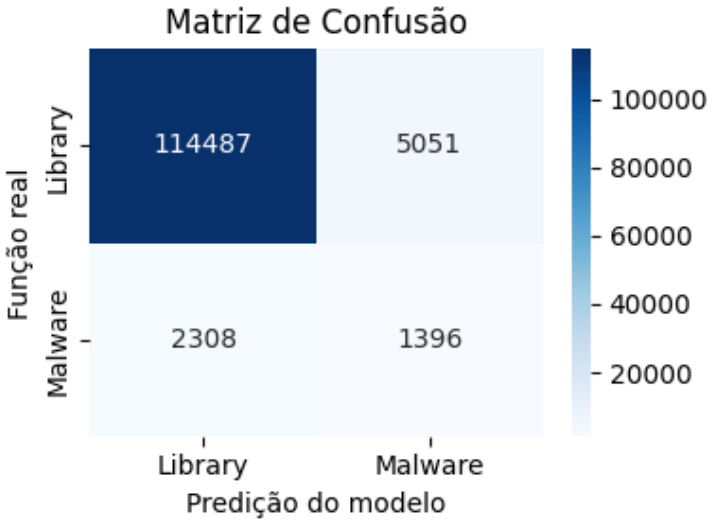

Figura 2. Matriz de Confusão - Classificação Baseada em Métricas

classificação. O modelo classificou corretamente 114.487 das 119.538 funções de biblioteca, mas cometeu 2.308 erros do tipo falso negativo, onde funções de malware foram erroneamente classificadas como benignas. Este tipo de erro acarreta consequências de segurança significativamente mais graves, pois funções maliciosas que deveriam ser objeto de estudo prioritário seriam potencialmente ignoradas pelo analista, representando um risco substancial de que ameaças reais não sejam detectadas. Em contrapartida, a baixa precisão na classe de malware reflete uma elevada taxa de falsos positivos, nos quais um número substancial de funções de biblioteca foi incorretamente classificado como malware. As consequências deste tipo de erro são primariamente operacionais, resultando em perda de tempo e recursos para o analista que investigaria código benigno.

A análise dos resultados demonstra que, embora a abordagem baseada em métricas de software seja computacionalmente eficiente, ela exibe uma capacidade limitada para discriminar funções maliciosas no contexto de um conjunto de dados desbalanceado. O modelo demonstra um forte viés em direção à classe majoritária (bibliotecas), resultando em um desempenho de detecção de malware que não é satisfatório para aplicações práticas de segurança.

Tabela 2. Resultados - Classificação Baseada em Métricas

	precision	**recall**	**f1-score**	**support**
Library	0.98	0.96	0.97	119538.0
Malware	0.22	0.38	0.28	3704.0
accuracy	0.94	0.94	0.94	0.94
macro avg	0.60	0.67	0.62	123242.0
weighted avg	0.96	0.94	0.95	123242.0

4. Ameaças à Validade

Nesta seção, nós discutimos as potenciais limitações do estudo que podem influenciar a interpretação e a generalização dos resultados. As ameaças são apresentadas a seguir.

Uma ameaça à validade do estudo está na dependência de ferramentas e na configuração do ambiente experimental. A abordagem baseada em métricas, por exemplo, utiliza a ferramenta comercial Understand da SciTools para a extração das características. Embora eficiente, esta dependência implica em potenciais custos de licença e vincula os resultados à implementação específica das métricas fornecidas pela ferramenta, o que pode dificultar a reprodutibilidade em ambientes que não disponham do mesmo recurso.

A estratégia de classificação semântica via embeddings, por sua vez, enfrenta duas limitações principais. Primeiramente, o parser desenvolvido para a extração de funções foi implementado para analisar apenas arquivos em linguagem C. Isso restringe a aplicabilidade do modelo a malwares escritos em outras linguagens. Em segundo lugar, o processo de análise semântica apresentou um custo computacional elevado, demandando entre 4 a 5 horas de processamento, em contraste com os 5 minutos da abordagem por métricas. Tal fator pode limitar a generalização dos resultados para cenários que exijam uma análise em larga escala e com maior celeridade.

5. Conclusão

Neste estudo, realizamos uma avaliação comparativa de duas abordagens distintas para a triagem de funções de malware: uma baseada em análise de métricas de software e outra em análise semântica (via Embeddings). Os resultados indicam um claro trade-off entre eficiência computacional e desempenho preditivo. A abordagem por análise de métricas estáticas de código demonstrou ser computacionalmente menos custosa. O tempo de processamento para esta abordagem foi de aproximadamente 5 minutos, em contraste com as 4 a 5 horas exigidas pela análise semântica, representando uma aceleração de 48 a 60 vezes. Uma vantagem adicional deste método é sua independência da linguagem de programação, permitindo a análise de todas as funções de um repositório sem a necessidade de desenvolver extratores específicos. Notavelmente, esta abordagem facilita a aplicação de técnicas de geração de dados sintéticos para o balanceamento do modelo, uma tarefa mais complexa na abordagem semântica devido à escassez de código-fonte de malware disponível.

Por outro lado, a abordagem por análise semântica apresentou uma precisão superior em suas predições, um resultado que pode ser parcialmente atribuído ao menor desbalanceamento de classes observado na configuração experimental utilizada. Para validar e expandir estes achados, sugerimos como trabalho futuro a utilização de uma base de dados de malware mais ampla e diversificada. O repositório VXUnderground [vx-underground 2021] apresenta-se como um candidato promissor, apesar dos desafios logísticos associados à descompactação e análise de seu vasto acervo. Por fim, os resultados apontam que, embora ambas as abordagens necessitem de maior refinamento, os modelos desenvolvidos demonstram uma capacidade de prototipagem considerável para aplicação em tarefas de campo. Tal aplicação poderia auxiliar analistas a realizar uma análise mais rápida e precisa de malwares, resultando em uma economia de tempo e acelerando o desenvolvimento de ferramentas para a mitigação de softwares maliciosos.

Referências

Admass, W. S., Munaye, Y. Y., and Diro, A. A. (2024). Cyber security: State of the art, challenges and future directions. *Cyber Security and Applications*, 2:100031.

Andrade, G., Cirilo, E., Durelli, V., Cafeo, B., and Adachi, E. (2020). Data-flow analysis heuristic for vulnerability detection on configurable systems. In *Anais do VIII Workshop de Visualização, Evolução e Manutenção de Software*, pages 25–32, Porto Alegre, RS, Brasil.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Ding, I. (2021). iot-malware: Iot malware dataset. `https://github.com/ifding/iot-malware`. Accessed: 2025-07-01.

Ebrahim, F. and Joy, M. (2023). Source code plagiarism detection with pre-trained model embeddings and automated machine learning. In Mitkov, R. and Angelova, G., editors, *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, pages 301–309, Varna, Bulgaria. INCOMA Ltd., Shoumen, Bulgaria.

Jia, Y., Yu, Z., and Hong, Z. (2024). Semantic aware-based instruction embedding for binary code similarity detection. *PLOS ONE*, 19(6).

Kathuria, P., Aggarwal, V., and Gupta, D. (2021). A comprehensive investigation of computer-based and mobile-based malware, their countermeasures, and various detection methods. *Computer Networks*, 195:108–157.

Li, M., Wang, W., Wang, P., Wang, S., Wu, D., Liu, J., Xue, R., and Huo, W. (2017). Libd: Scalable and precise third-party library detection in android markets. In *Proceedings of the 39th IEEE/ACM International Conference on Software Engineering (ICSE 2017)*, pages 335–346. IEEE.

Shalaginov, A., Banin, S., Dehghantanha, A., and Franke, K. (2018). Machine learning aided static malware analysis: A survey and tutorial. *Computers Security*, 80:41–60.

The OpenBSD Project (2025). Openbsd source tree. `https://github.com/openbsd/src`. Accessed: 2025-07-01.

Votipka, D., Rabin, S. M., Micinski, K., Foster, J. S., and Mazurek, M. M. (2020). An observational investigation of reverse engineers' processes. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC'20.

vx-underground (2021). Malware source code collection. Accessed: 2025-07-04.