ISSN: 1983 7402 ITA, 27 a 29 SET 2022

Classificação de malware utilizando ASM2VEC

Rafael Oliveira da Rocha[1], Lourenço Pereira Jr.[1] e Idilio Drago[2]
[1]Instituto Tecnológico de Aeronáutica, São José dos Campos/SP - Brasil
[2]Politecnico di Torino, Turin - Itália

***Resumo*— A ocorrência de malware em sistemas é cada vez maior dada a variedade de programas maliciosos e ao maior volume de dados trafegando na Internet. A infecção é danosa para os usuários e organizações e, por isso, os sistemas devem buscar técnicas mais eficientes para a detecção destas ameaças. A abordagem deste trabalho foi estudar a identificação do malware pelo seu código assembly por meio da transformação destas instruções para um vetor numérico, por meio do algoritmo ASM2VEC.**

***Palavras-Chave*— Malware,ASM2VEC.**

I. Introdução

Malware é um termo genérico para qualquer tipo de programa malicioso, projetado para prejudicar usuários, organizações, sistemas de telecomunicações e computadores. Ao infectar um sistema, o programa pode apagar ou corromper arquivos, roubar informações relevantes (senhas, dados bancários, etc) [1] e [2]. A contaminação de um sistema por malware é algo difícil de ser evitada, dado o crescimento exponencial do uso da Internet no mundo [3]. Dessa forma, a proteção deve ser no nível das contramedidas, visando minimizar a possibilidade de infecção.

Para tanto, são necessárias a detecção do malware no sistema e a sua remoção, tarefas realizas por um antivírus ou de forma manual [1]. Esta identificação pode ser obtida por meio de comparação de assinaturas ou por aprendizado de máquina. No primeiro caso, há muito esforço humano para a criação das assinaturas e o resultado possui baixa eficácia frente a novas variantes, pois esses programas evoluíram usando técnicas antidetecção, apresentando códigos não estruturalmente similares [2]. Já a segunda abordagem é considerada mais apta para a detecção e classificação de malwares, e analistas de segurança têm identificado e utilizado diversas características de malware para realizar classificação via aprendizado de máquina [4]. O rol de tais características já experimentadas na detecção de malware é vasto, abrangendo strings em texto claro no código, sequência de bytes do código, código assembly, chamadas de sistema, grafos de fluxo de controle, funções, tamanho do arquivo, código de saída, consumo de tempo de execução, entropia, empacotamento no código e número de instruções dinâmicas/estáticas. Entretanto, devido ao fato de desenvolvedores de malware modificarem constantemente seus códigos para evadir-se das detecções, a classificação por aprendizado de máquina continua sendo um desafio [2].

A abordagem proposta para o presente estudo é a classificação do malware pelo seu código assembly e utilizando o algoritmo ASM2VEC [5], que tem o potencial de atingir um resultado robusto frente às tentativas de evasão dos desenvolvedores de malware [2]

II. Revisão Bibliográfica

A. Malware

Existem várias categorias de malware que se diferem por sua forma de atuação [1]. Entre essas categoriasm podemos citar as dos programas que podem encriptar e travar computadores (Ransomware), que se replicam de forma independente e se espalham pela rede (Worm), que que roubam informações do usuário (Trojan e Spywire), que permitem que o computador do usuário seja controlado externamente por um hacker (Backdoor), que forçam a apresentação de propagandas e janelas de popups (Adware), entre outras.

Malware estão em constante evolução, gerando novas variantes de uma mesma categoria, para evitar que sejam detectados pelo sistema a ser infectado. A evolução dos programas maliciosos, acabam afetando o desempenho dos modelos de classificação, que acabam perdendo sua eficácia com o passar do tempo, em um fenômeno conhecido por drifting [6] e [7].

Além disso, desenvolvedores de malware utilizam-se de técnicas de ofuscação para subverter a detecção e classificação de códigos maliciosos. Entre tais técnicas, pode-se citar a introdução de código morto (como instruções NOP), transposição de código, substituições de instruções, entre outras.

Outro problema que afeta a classificação de malware é a suscetibilidade de modelos baseados em aprendizado de máquina a ataques adversários (adversarial attacks). Tais ataques podem ocorrer tanto na fase de treino quanto na fase de treino (poisoning attack) quanto na fase de teste (evasion attacks) e visam perturbar/manipular as amostras de malwware de modo a ludibriar o modelo de detecção/classificação[8].

Nesse contexto, o ASM2VEC, que é uma técnica para identificação de clones de função assembly, mostra-se promissor para a tarefa de gerar um modelo de classificação de malware mais robusto. Isso se deve ao fato de que o ASM2VEC tem a capacidade de detectar clones semânticos, ou seja, consegue identificar códigos que, ainda que tenham estruturas diferentes, tenham a mesma funcionalidade. Em testes realizados, o ASM2VEC foi capaz de identificar como clones funções com estruturas consideravelmente diferentes (como aquelas geradas por compiladores distintos ou, até mesmo, ofuscadas)[5].

B. Trabalhos Relacionados

Na literatura de classificação de malware, existem abordagens que utilizam os diversas fontes distintas como características para detecção. Além disso, diversas técnicas de aprendizado de máquina já foram experimentadas para realizar a classificação, tais como: Naive Bayes classifier (NBC), rule-based classifier, decision tree (DT), K-nearest neighbors (K-NN), Bayesian Network, Neural Network (NN), Convolution

Neural Network (CNN), Random Forest (RF), Hidden Markov Models (HMM) e Support Vector Machine (SVM) [2].

Em relação à fonte de dados, o aprendizado de máquina usando o código assembly possui algumas vantagens, como sua facilidade em ser entendido pelo analista e a possibilidade de, por meio da extração dinâmica, obter-se as instruções do código malicioso que estejam criptografadas no código estático. Entretanto, existem alguns desafios para essa abordagem, como sua fragilidade à inserção de código morto e transposição de operações [2].

Diversas abordagens de classificação de malware já foram utilizadas tendo como fonte as instruções assembly, utilizando o Word2Vec, one-hot e FastText [9], [10] e [10] e obtiveram uma acurácia maior de 96%. Entretanto, apesar da boa eficácia, esses algoritmos não se mostraram robustos contra as técnicas de antidetecção dos malwares [2].

Nesse contexto, o ASM2VEC representa uma nova proposta de estudo promissora, pois, teoricamente, pode apresentar resultados mais robustos contra variações dos programas maliciosos, dada a sua capacidade em identificar códigos similares.

C. ASM2VEC

O Word2Vec é uma técnica baseada em rede neural na qual a rede é treinada com as palavras de um texto. A partir dessa palavras é gerado um vetor numérico que captura uma relação representativa das palavras da base de treinamento [9]. Já o algoritmo ASM2VEC, que é inspirado no Word2Vec, tem a capacidade de aprender uma representação vetorial a partir de um código assembly [2].

Desta forma, ASM2VEC treina uma rede neural a partir de um repositório de arquivos .asm (código Assembly) e tenta entender suas dimensões latentes subjacentes [5]. Isto gera um modelo que representa o contexto dos códigos assembly existentes no repositório. A partir do modelo aprendido, um novo arquivo é convertido para um vetor e então sua similaridade pode ser comparada com a dos vetores existentes no repositório, conforme ilustrado na Figura 1.

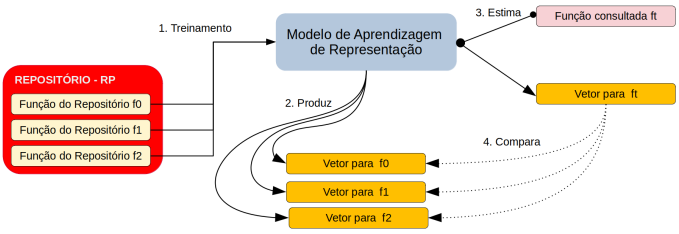

Fig. 1. Fluxo de dados do ASM2VEC, adaptado de [5].

De acordo com a Figura 1, no passo 1, é montado um repositório de funções assembly que serão utilizadas para treinar o modelo. No passo seguinte (2), são produzidos os vetores numéricos que representam as funções do repositório. Já no passo 3, para uma função alvo (ft) que se deseja verificar, é gerado o seu vetor a partir do modelo criado no treinamento e, por fim, no passo 4, o vetor da função alvo é comparado com os vetores da função do repositório e calculado o respectivo nível de semelhança.

Desta maneira o ASM2VEC é capaz de identificar se um código dado é um clone do tipo 4 (funções assembly que podem parecer sintaticamente diferentes, mas que compartilham lógica funcional semelhante em seus códigos) de funções conhecidas, ou seja, se há alguma similaridade semântica na implementação das funções. O ASM2VEC demonstrou eficiência na identificação de arquivos semelhantes apesar da utilização de compiladores com diferentes níveis de otimização, ofuscação e ruído [5].

Teoricamente, a aplicação do ASM2VEC seria capaz de identificar variantes desconhecidas do malware [2], mostrando-se uma técnica robusta para a classificação de programas maliciosos.

III. Materiais e Métodos

A. Conjunto de Dados

A base de dados utilizada no trabalho foi a BIG 2015, com um total de 500 GB, oriunda da competição *Microsoft Malware Classification Challenge*[10]. O conjunto de dados contém nove tipos diferentes de famílias de malware, sendo 1- Ramnit, 2- Lollipop, 3-Kelihos_ver3, 4-Vundo, 5-Simda, 6-Tracur, 7-Kelihos_ver1, Obfuscator.ACV e 9-Gatak.

B. Técnicas

No trabalho, foi utilizado o algoritmo ASM2VEC. Para tal, foi empregada uma implementação do ASM2VEC disponível no Github [11].

Para realizar a classificação de malware, foram realizados os passos do fluxograma ilustrado na Figura 2 e observando os passos a seguir:

- Passo 1: inicialmente o dataset é dividido em nove repositórios, um correspondente a cada família de malware;
- Passo 2: em seguida, é gerado um modelo para cada classe e calculado os vetores das funções de cada repositório;
- Passo 3: para um código a ser testado, é calculado seu vetor para cada modelo e verificada a sua semelhança (via similaridade por cosseno) com as funções dos respectivos repositórios;
- Passo 4: é comparado o nível de semelhança do código testado com cada um dos repositórios. A maior semelhança define o rótulo que o código testado recebe.

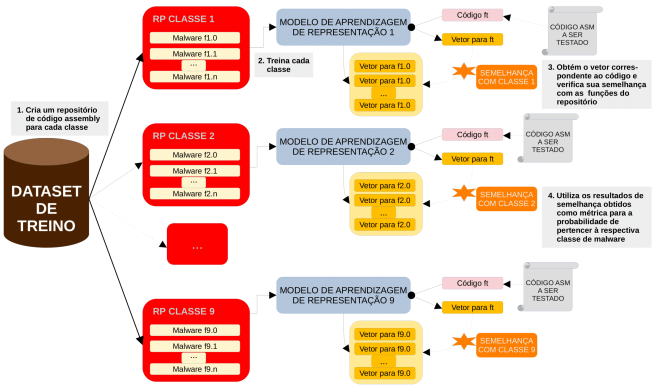

Fig. 2. Fluxo de dados para classificação baseado no ASM2VEC.

Para aplicar o ASM2VEC, foi tomado por base o código asm2vec-pytorch, que é uma implementação não oficial de asm2vec usando pytorch [11].

IV. Experimentos e Resultados

A. Exploração dos Dados

A base de dados de treinamento do BIG 2015 (train.7z) do BIG 2015 possui as classes rotuladas no arquivo trainLabels.csv e, portanto, foi dividida na metade sendo uma parte utilizada para o treinamento e outra parte utilizada como base de dados de teste. O total de objetos de treinamento utilizado foi de 5.380 e para teste foi de 5.380.

A base de dados pode ser representada como uma tabela atributo-valor no formato apresentado na Figura 3.

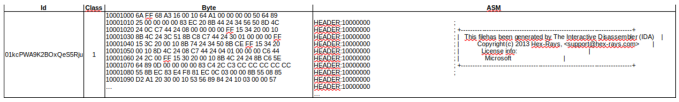

Fig. 3. Tabela ilustrativa de atributo-valor com um objeto da base de dados.

Cada objeto da base de dados é formado por um arquivo .asm e outro arquivo .bytes de mesmo nome.

No arquivo .byte o malware é representado por seu código binário em formato hexadecimal, sem o cabeçalho para garantir a esterilidade, além de metadados contendo várias informações (chamadas de função; strings; etc) extraídas do binário pela ferramenta IDA. No arquivo .asm, além do código hexadecimal do arquivo são apresentadas as instruções assembly correspondentes.

A distribuição das classes de malware não é balanceada, conforme apresentado na Figura 4.

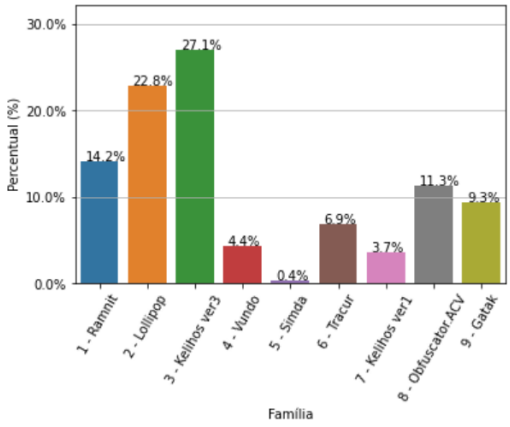

Fig. 4. Percentual das classes de malware no BIG 2015.

A Tabela I contém os atributos existentes na base de dados. Conforme pode ser observado, todas as features são qualitativas nominais e o atributo alvo tem formato de numérico, que representa a família a qual o malware pertence.

TABELA I

Classificação dos atributos.

Atributo	Descrição	Classificação
Id	Código hash que representa o objeto do dataset, servindo como um índice de banco de dados	Qualitativo nominal
Class	Classificação da família do malware (Atributo alvo)	Quantitativo discreto
Byte	Sequência do código binário do arquivo em formato hexadecimal	Qualitativo nominal
ASM	Código do arquivo em formato hexadecimal e a correspondente instrução assembly	Qualitativo nominal

B. Pré-processamento dos Dados

A fase de pré-processamento tem o objetivo de melhorar a qualidade dos dados do dataset, de modo a viabilizar o aprendizado de máquina. Na abordagem de classificação proposta neste trabalho, os arquivos .byte não serão utilizados, podendo ser removidos.

Assim, apenas os arquivos .asm são utilizados no aprendizado de máquina, sendo inicialmente separados em repositórios de acordo com o rótulo da classe.

Os arquivos .asm da base de dados contêm muitas informações que não são instruções assembly, como os bytes correspondentes ao endereço de memória e linhas de cabeçalho. Desta forma, foi necessário realizar alguns ajustes no BIG2015, conforme explicado a seguir:

1) **Eliminação de objetos**: alguns arquivos .asm do dataset não apresentavam nenhuma informação sobre instruções assembly, conforme ilustra a Figura 5. Como o ASM2VEC necessita de funções com instruções assembly, os arquivos que não tinham essa característica foram eliminados. No total, foram eliminados 147, restando 5.362 objetos para o treinamento e 5.359 para a realização dos testes.

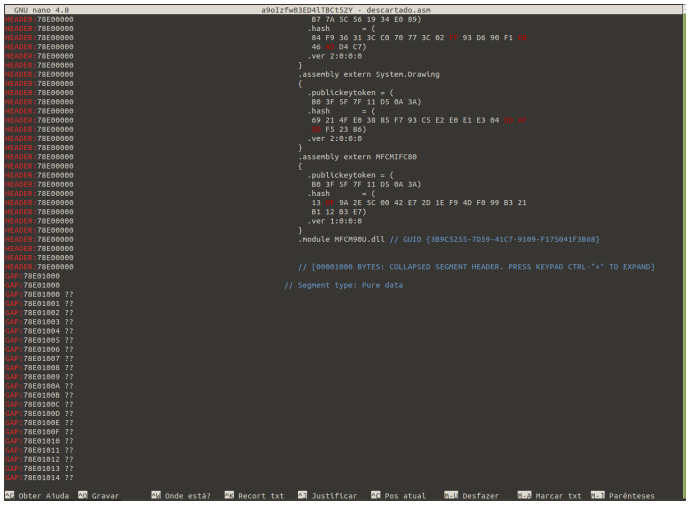

Fig. 5. Exemplo do conteúdo de objeto descartado: arquivo a9oIzfw03ED4lTBCt52Y.asm.

2) **Formatação dos arquivos .asm**: a implementação do ASM2VEC necessita receber como entrada um formato de arquivo bem definido, composto por nome da função, seguido de dois pontos e, abaixo, as respectivas instruções. Para formatar o dataset no padrão necessário, foi criado um script em Shell que recebe os arquivos .asm do dataset, filtra somente as linhas que contenham nome de função ou suas instruções e ajusta o espaçamento.
3) **Cálculo prévio dos vetores**: a implementação do ASM2VEC utilizada neste trabalho [11] cria repositórios de funções assembly e realiza o cálculo dos respectivos vetores no momento da comparação com uma função alvo. Ou seja, o algoritmo efetua cálculo de vetores para todas as funções do repositório a cada função alvo a ser testada. Como o cálculo de vetor de função é computacionalmente oneroso e o BIG2015 possui milhares de objetos, essa abordagem é potencialmente inviável de produzir resultados tempestivos.

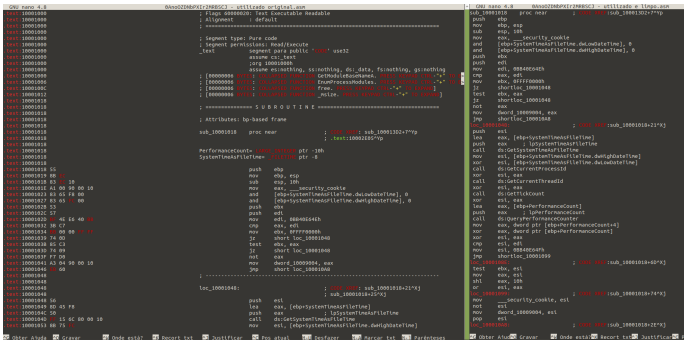

Fig. 6. Exemplo de conteúdos original e limpo de arquivo .asm.

Dessa forma, foi feita uma adaptação no código para que os vetores das funções dos repositórios fossem previamente calculados.

C. Geração do modelo de aprendizado de máquina e classificação

Criados, para cada classe, os respectivos modelos e repositórios de vetores, foi possível utilizar similaridade por cosseno para comparar a semelhança dos vetores da base de dados de teste com os vetores dos repositórios. O maior valor de semelhança foi utilizado definir a qual classe o objeto da base de testes pertence.

Feito o processo de classificação supracitado, obteve-se a matriz de confusão apresentada na Figura 7. As métricas da classificação por classe obtidas são apresentados na Tabela II.

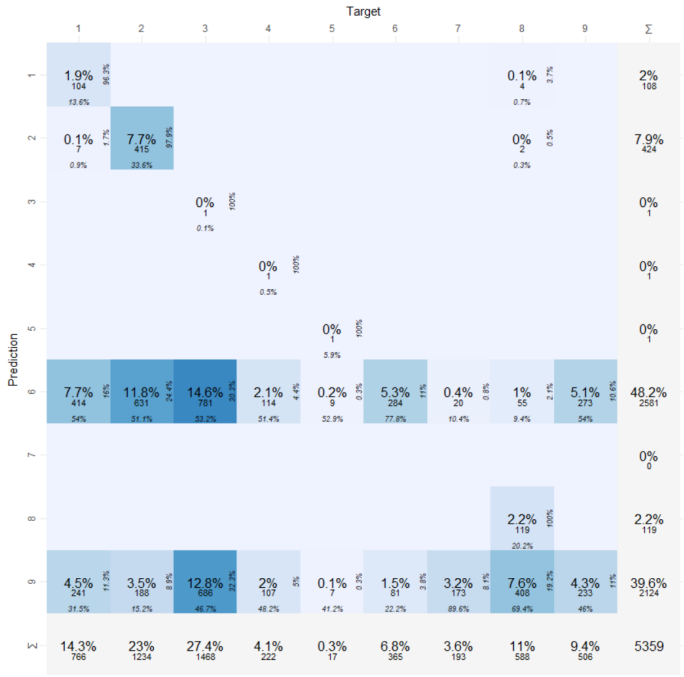

Fig. 7. Matriz de confusão.

V. Discussão

Com o resultado da predição, verifica-se que o algoritmo utilizado teve um baixo desempenho preditivo. Observa-se que houve uma tendência a classificar os objetos como pertencentes às classes 6 (48,2% do total das rotulações) e 9 (39,6% das rotulações), mesmo essas classes não sendo as majoritárias (representando 2,93% e 10,00% do total de objetos, respectivamente).

Apesar da alta tendência de classificação nas classes supracitadas, o valor para suas respectivas sensibilidades não chegou a 80% (77,8% para a classe 6 e 46,0% para a classe 9).

Outros três resultados que chamam a atenção são os valores quase nulos de sensibilidade para as classes 3 (0,1%) , 4 (0,5%) e 7 (nulo). Sendo que a classe 3 é a majoritária no que tange ao número de objetos do datase, o que representa um indício de que o viés de classificação para as classes 6 e 9 não se deve a desbalancamento da base.

A Tabela III apresenta o número de objetos em cada classe e o número de função em cada classe. Como pode ser observado, as classes 6 e 9, que o algoritmo teve preferência por classificar, não são majoritárias, tanto em relação à quantidade de objetos quanto em relação ao número de funções existentes no dataset. Tal fato, a priori, descarta alguma relação quantitativa de desbalanceamento entre classes com o baixo desempenho preditivo que o modelo apresentou.

TABELA II
MÉTRICAS DA CLASSIFICAÇÃO DA PREDIÇÃO DE APRENDIZADO DE MÁQUINA.

Classe	Sensibilidade (%)	Especificidade (%)	F1 (%)	Predição positiva (%)	Predição negativa (%)	Taxa de detecção (%)
1	13,6	99,9	23,8	96,3	87,4	1,9
2	33,6	99,7	50,0	97,9	83,4	7,7
3	0,1	100	0,1	100	72,6	0,0
4	0,5	100	0,9	100	95,9	0,0
5	5,9	100	11,1	100	99,7	0,0
6	77,8	54,0	19,2	11,0	97,0	5,3
7	0,0	100	NaN	NaN	96,4	0,0
8	20,2	100	33,6	100	91,1	2,2
9	46,0	61,0	17,7	11,0	91,6	4,3

TABELA III
NÚMERO DE OBJETOS E DE SUBFUNÇÕES DE CADA CLASSE.

Classe	Número de objetos	Número de sub-rotinas
1	766	669.523
2	1.234	124.463
3	1.468	87.800
4	222	3.357
5	17	397
6	365	26.299
7	193	17.642
8	588	54.853
9	506	36.418

Diante dos dados, uma hipótese para a causa do resultado obtido é a de que existam sub-rotinas comuns entre as classes de malware, que desempenham funções iguais ou semelhantes (por exemplo: compactar arquivos, realizar cópias do próprio código, baixar um arquivo, etc), e que a quantidade dessas funções comuns seja significante comparada à das funções maliciosas específicas das classes.

Para identificar tais funções comuns, seria necessário aumentar a granularidade dos arquivos da base de treino, gerando um arquivo para cada sub-rotina dos malwares. Com essa subdivisão, seria possível identificar similaridades entre os repositórios e remover tais sub-rotinas para, depois, realizar novamente o teste de classificação.

Além disso, a escolha do critério para a definição da classe a qual o objeto em teste pertence pode também afetar o desempenho do modelo. No trabalho realizado, o critério utilizado foi o de máxima similaridade: para um objeto a ser testado, verifica-se o valor máximo de similaridade com os objetos do repositório de uma classe, representando esse valor a probabilidade do objeto pertencer à respectiva classe. Estudos podem ser feitos no sentido de identificar outros critérios que possam produzir melhores resultados de classificação em vez de utilizar a máxima similaridade (ex: média das similaridades de cada repositório; média das n maiores similaridades; mediana das similirades; etc).

VI. Conclusão

A abordagem de identificação de malware em estudo, que utiliza o algoritmo ASM2VEC, apresentou um baixo poder preditivo. Uma hipótese para o baixo desempenho é a que existam de funções que são comuns a mais de uma classe e em número superior ao das funções que executam as ações maliciosas específicas da classe. Tal hipótese deve ser investigada e, em sendo confirmada, verificado o desempenho após a eliminação de tais funções.

Desta forma, uma proposta para melhorar o desempenho da classificação é a inclusão de uma etapa de filtragem adicional na etapa de pré-processamento para eliminar funções similares antes da geração dos modelos ASM2VEC.

A escolha do critério utilizado para classificação pode também influenciar no resultado do modelo de classificação. No presente trabalho, o critério utilizado foi o valor máximo de similaridade obtido ao se comparar um código alvo com cada código do repositório. Nesse sentido, outros estudos podem ser realizados no sentido de propor métricas diversas que eventualmente possam melhorar o desempenho do algoritmo.

Por fim, cabe ressaltar ainda que trabalhos recentes de classificação de malware utilizam diversas técnicas para a identificação e classificação de trechos de código malicioso, em vez de classificar o código como um todo [12] e [13].Na mesma linha, estudos futuros podem ser realizados no sentido de avaliar o desempenho do ASM2VEC na identificação de similaridade de trechos de códigos maliciosos.

Referências

[1] M. A. H. Saeed, "Malware in computer systems: Problems and solutions," *IJID (International Journal on Informatics for Development)*, vol. 9, p. 1, 4 2020.

[2] A. Abusitta, M. Q. Li, and B. C. Fung, "Malware classification and composition analysis: A survey of recent developments," *Journal of Information Security and Applications*, vol. 59, 6 2021.

[3] M. Q. Li, B. C. Fung, P. Charland, and S. H. Ding, "A novel and dedicated machine learning model for malware classification." SciTePress, 2021, pp. 617–628.

[4] D. Gibert, C. Mateu, and J. Planes, "Hydra: A multimodal deep learning framework for malware classification," *Computers and Security*, vol. 95, 8 2020.

[5] S. H. Ding, B. C. Fung, and P. Charland, "Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization," vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., 5 2019, pp. 472–489.

[6] Z. Kan, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Investigating labelless drift adaptation for malware detection," in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 123–134. [Online]. Available: https://doi.org/10.1145/3474369.3486873

[7] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending transcend: Revisiting malware classification in the presence of concept drift," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 805–823.

[8] D. Li, Q. Li, Y. F. Ye, and S. Xu, "Arms race in adversarial malware detection: A survey," *ACM Comput. Surv.*, vol. 55, no. 1, nov 2021. [Online]. Available: https://doi.org/10.1145/3484491

[9] A. S. Kale, F. D. Troia, and M. Stamp, "Malware classification with word embedding features." SciTePress, 2021, pp. 733–742.

[10] Y. Sung, S. Jang, Y. S. Jeong, and J. H. J. J. Park, "Malware classification algorithm using advanced word2vec-based bi-lstm for ground control stations," *Computer Communications*, vol. 153, pp. 342–348, 3 2020.

[11] Oalieno, "Unofficial implementation of asm2vec using pytorch (with gpu acceleration). endereço eletrônico: https://github.com/oalieno/asm2vec-pytorch/tree/master/scripts. acessado em abril de 2022," 2021. [Online]. Available: https://github.com/oalieno/asm2vec-pytorch/tree/master/scripts

[12] C. Molloy, P. Charland, S. H. H. Ding, and B. C. M. Fung, "Jarv1s: Phenotype clone search for rapid zero-day malware triage and functional decomposition for cyber threat intelligence," in *2022 14th International Conference on Cyber Conflict: Keep Moving! (CyCon)*, vol. 700, 2022, pp. 385–403.

[13] E. Downing, Y. Mirsky, K. Park, and W. Lee, "DeepReflect: Discovering malicious functionality through binary reconstruction," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3469–3486. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/downing