Bulletin Of High Technology N 3 (31) 2024.-pp. 77-83.

UDC - 004.725:004.852

MALWARE OBFUSCATION MODEL USING MACHINE LEARNING

Timur V. Jamgharyan
National Polytechnical University of Armenia
105 Teryan st, 0009, Yerevan
e-mail: t.jamharyan@ polytechnic.am
ORCID ID: 0000-0002-9661-1468
Republic of Armenia

Artak A. Khemchyan
National Polytechnical University of Armenia
105 Teryan st, 0009, Yerevan
e-mail: a.khemchyan@polytechnic.am
ORCID ID: 0009-0008-3271-1903
Republic of Armenia

https://doi.org/10.56243/18294898-2024.3-77

Abstract

The paper presents the results of a research of the use of a generative-adversarial network for software obfuscation. The comparison was carried out with deterministic obfuscators Dotfuscator CE, Net Reactor, ProGuard. The research of changes in the source code of the software athena, abc, cheeba, dyre, december_3, engrat, surtr, stasi, otario, dm, v-sign, tequila, flip, grum, mimikatz was carried out using the IDA Pro disassembly tool. Detection of obfuscated malware was carried out based on the piecewise context fuzzy hashing method. The research was conducted to use of a generative-adversarial network as an obfuscator on pre-specified datasets. Simulation of the developed method was carried out in the Hyper-V virtual environment.

Keywords: obfuscation, reverse engineering, data flow, generative-adversarial network, machine learning, *ssdeep, IDA Pro*.

Introduction

The use of machine learning (ML) to detect malware in network infrastructure (NI) has taken attack and defense tools to a new level. One of the methods that makes it difficult to analyze source code is obfuscation[1]. There are a large number of obfuscator algorithms, but most of them are based on two fundamental algorithms: *Kohlberg's algorithm and ChenxiWang's algorithm*. Based on them, various obfuscating software have been developed (*Dotfuscator CE, Net Reactor, ProGuard, etc*.) that perform various types of obfuscation

[1]Obfuscation is the reduction of the source text or executable code of a program to a form that preserves its functionality, but complicates analysis, understanding of operating algorithms and modification during decompilation [1].

Bulletin Of High Technology N 3 (31) 2024.-pp. 77-83.

(lexical, preventive, data structures, data stream).But their use is limited by their determinism, since the obfuscation algorithms embedded in various obfuscation software are deterministic. Accordingly, the task of researching and developing an obfuscator that changes the set of obfuscation parameters according to an algorithm containing a stochastic element becomes urgent. Various researchers are trying to solve this problem [2-5], including using ML methods [6-8], solving it within the given limitations. This research investigates the use of a neural network for data stream[2]obfuscation. The novelty of the research lies in the obfuscation of the software data stream by adding fragments of «unreachable code»[3] generated by a neural network.

Conflict Setting

Explore a model for using a neural network to obfuscate software source code.

Discussion

A generative-adversarial network was used as a neural network used for software obfuscation in this research. Instances of data sets generated by the generative-adversarial network were entered into the software source code. Adding components of «unreachable code», in contrast to false code, allows you to reduce the boundary between «true code» and false code when reverse engineering software. In this case, versions were added that were generated by a generative-adversarial network based on previous versions of malware, as well as open source software that was not malicious. The experiment was carried out using malware *athena, abc, cheeba, dyre, december_3, engrat, surtr, stasi, otario, dm, v-sign, tequila, flip, grum, mimikatz* of various versions.

Experimental procedures

The Hyper-V role is installed in the Windows Server 2016 Standard operating system environment. This virtual environment has Ubuntu v20.04 installed with a pre-configured generative adversarial network. The generative adversarial network was trained on the basis of data sets obtained from the source code, software under study and sources [11,12].Training was carried out using the «unsupervised»[4] method. To protect against static analysis of obfuscated software, duplicate components of executable code were additionally added to it based on code from the open source of various applications. The signature difference of the generated «unreachable code» for malware obfuscation was calculated using edit distance. The research was carried out with a piecewise contextual fuzzy hashing step of 32, 64, 128, 256 bytes.

Research Results

Fig. 1-4 shows visualizations of the malware detection results of *athena, abc, cheeba, dyre, surtr, stasi, v-sign* obfuscated using a neural network at different training epochs. The

[2]Data stream obfuscation is a type of transformation aimed at changing the order of execution of program blocks [9].

[3] «Unreachable code» is a part of the program code that cannot be executed because it is unreachable in the control flow graph [10].

[4]Unsupervised learning is one of the methods of machine learning in which the system under test spontaneously learns to perform a given task without intervention from the experimenter.

Bulletin Of High Technology N 3 (31) 2024.-pp. 77-83.

piecewise context fuzzy hashing method was used as a research method, and ssdeep software and resources were used as a research tool [13,14]. In Fig. 5 shows the *surtr* software subjected to the reverse engineering procedure after its obfuscation using a neural network.

Tab. 1 presents some results comparing deterministic obfuscators and an obfuscator using a generative-adversarial network when obfuscating a software data stream.

Table 1

Comparison of obfuscation parameters of the software under research

Obfuscator/parameter	Dotfuscator CE	Net Reactor	ProGuard	Research method
Feature map[5]	48x48	32x32	--	Determined by the number of layers
Bytecode increase (%)	11.4	7.3	8.2	3.1-4.2
Structural similarity[6] (%)	54.3	58.2	61.2	78.3-83.6

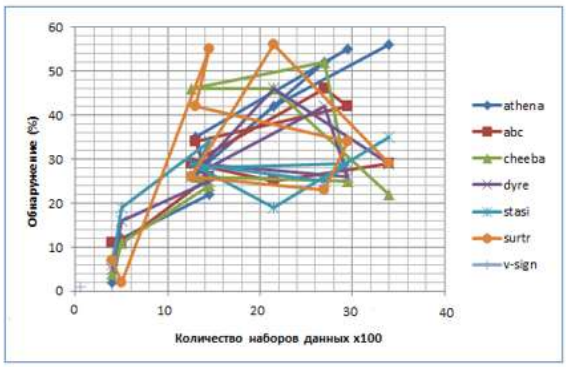

Fig. 1. Visualization of the results of detection of malware *athena, abc, cheeba, dyre, surtr, stasi,, v-sign* obfuscated using a neural network(I learning epoch).

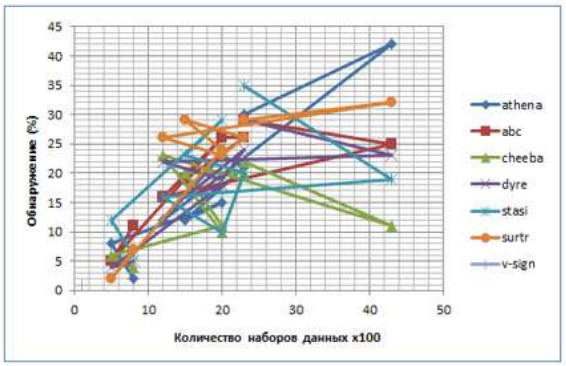

Fig. 2. Visualization of the results of detection of malware *athena, abc, cheeba, dyre, surtr, stasi,, v-sign* obfuscated using a neural network(II learning epoch).

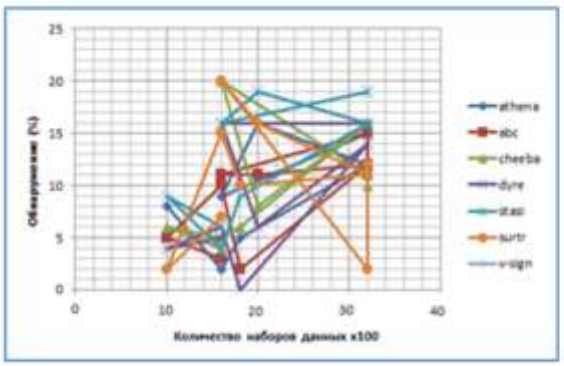

Fig. 3. Visualization of the results of detection of malware *athena, abc, cheeba, dyre, surtr, stasi,, v-sign* obfuscated using a neural network (III learning epoch, II iteration).

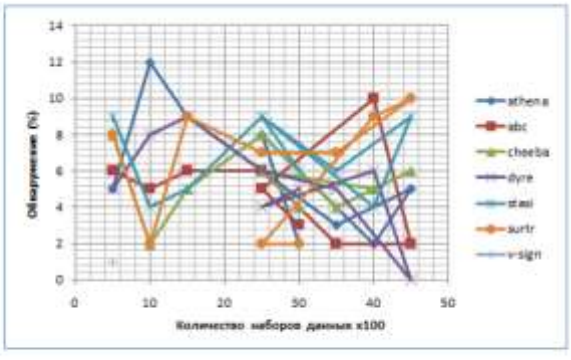

Fig. 4. Visualization of the results of detection of malware *athena, abc, cheeba, dyre, surtr, stasi,, v-sign* obfuscated using a neural network (III learning epoch, III iteration).

[5]The feature map is represented by a Gram matrix.
[6] Determined by piecewise contextual fuzzy hashing using the ssdeep tool

Bulletin Of High Technology N 3 (31) 2024.-pp. 77-83.

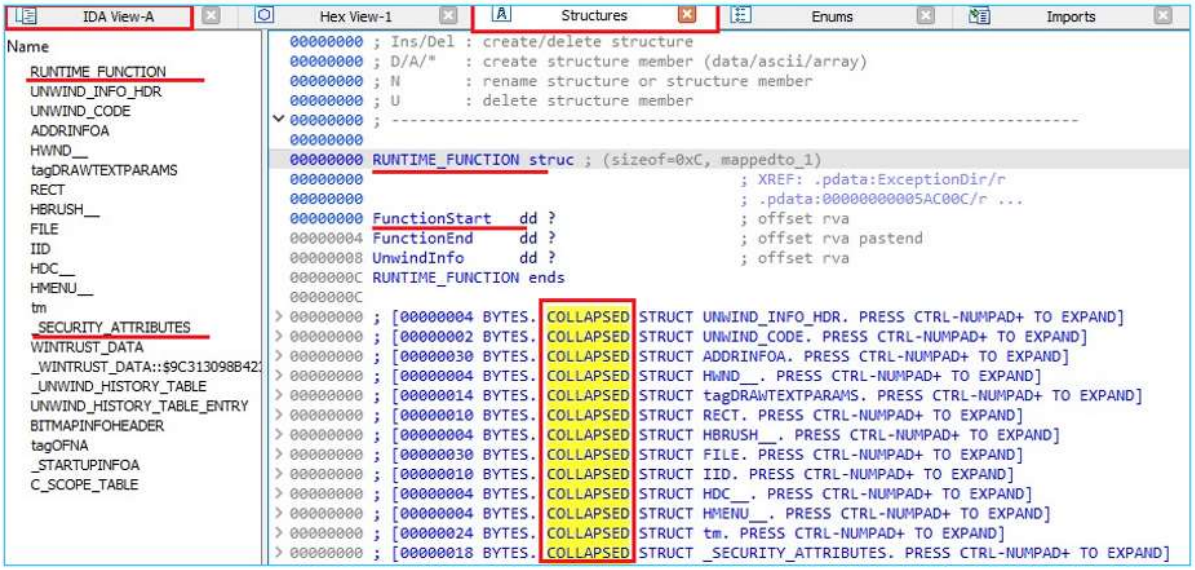

Fig. 5. Reverse engineering obfuscated *surtr* software code using the IDA Pro tool.

The use of a generative-adversarial network to generate data in the program control flow allows increasing the level of software obfuscation. In all cases, the use of a neural network to obfuscate the software source code is justified during its subsequent detailed examination and compilation. Each instance of obfuscated software, with equal versions, differs from other instances due to the stochastic element included in the neural network, which reduces the likelihood of its being opened using reverse engineering methods using static analysis. Detectability by deterministic means is also very low due (5÷7)% to the large number of false positives. For deterministic systems under equal conditions, the detectability is (14÷17)%.

The use of a generative-adversarial network for software obfuscation creates several challenges:

• Obfuscation of constants characteristic of certain algorithms leads to the inoperability of the entire code (Fig. 5).

• Non-obfuscation of algorithmic constants leads to the definition of the algorithm and, accordingly, disclosure of the software.

In all cases, the use of a generative-adversarial network to obfuscate a data stream is superior to deterministic obfuscators, while simultaneously requiring more computing resources. Also, as the training epoch increases, the number of false positives increases, which can indirectly unmask obfuscated software.

Conclusion

The paper discusses the possibility of using a generative adversarial network to obfuscate malware by adding «unreachable code» to the data stream of executable software. Static analysis of obfuscated software using a generative adversarial network showed a difference in instances of obfuscated software *athena*, *abc*, *cheeba*, *dyre*, *december_3*, *engrat*, *surtr*, *stasi*, *otario*, *dm*, *v-sign*, *tequila*, *flip*, *grum*, *mimikatz* with identical versions. When comparing obfuscated software with the use of deterministic obfuscators (*Dotfuscator CE*,

Bulletin Of High Technology N 3 (31) 2024.-pp. 77-83.

Net Reactor, ProGuard) in terms of detectability using the piecewise context fuzzy hashing method, the use of a neural network is more effective by (18÷24)%. At the same time, some problems inherent in systems with machine learning arise. Based on the research, some versions of obfuscated software were cataloged and an intrusion detection system with ML was configured in more detail. The full results of the study are presented in [15].

References

1. J. Aayush, H. Lin, A. Sahai, «Indistinguishability Obfuscation from Well-Founded Assumptions», https://eprint.iacr.org/2020/1003
2. C. Shangtong et al, «WASMixer: Binary Obfuscation for WebAssembly», https://doi.org/10.48550/arXiv.2308.03123
3. A. Durbet et al, «On the Leakage of Fuzzy Matchers», http://dx.doi.org/10.2139/ssrn.4592816
4. Z. Abideen, et al, «An Overview of FPGA-inspired Obfuscation Techniques», https://doi.org/10.48550/arXiv.2305.15999
5. T. Zhao, H. Hu, C. Lu, «Unveiling the Role of Message Passing in Dual-Privacy Preservation on *GNN*», https://doi.org/10.48550/arXiv.2308.13513
6. *N. Varnovsky, V. Zakharov, N. Kuzyurin, A. Shokurov, «The current state of research in the field of program obfuscation: determining the resistance of obfuscation»,* https://doi.org/10.15514/ISPRAS-2014-26(3)-9
7. A. Mazin, D. Shchelkunov, «Research and development of a new obfuscation method» // Bulletin of MSTUim. N.E. Bauman. Series «Instrument making». 2009. No. 2 https://cyberleninka.ru/article/n/issledovanie-i-razrabotka-novogo-metoda-obfuskatsii
8. *J. Diaz, A. Garcia «Comparison of machine learning models applied on anonymized data with different techniques»,* https://arxiv.org/abs/2305.07415
9. *S. K. Debray, «Compiler techniques for code compaction»,* https://doi.org/10.1145/349214.349233
10. *R. Mulyukov, A. Borodin, «Using unreachable code analysis in a static analyzer to find errors in the source code of programs», Proceedings of the Institute of System Programming RAS, 2016, 28 (5), 145-158.*
11. *Malware Bazaar Database,* official download page**.** *[Online]. Available* https://bazaar.abuse.ch/browse/
12. *Malware Bazaar Database,* official download page**.** [Online]. Available http://vxvault.net/ViriList.php , the resource is available on 20.04.2024.
13. Official page of the malware checking service, https://www.virustotal.com ,the resource is available on 20.04.2024.
14. Official page of the malware checking service, https://metadefender.opswat.com/All research results. https://github.com/T-JN , the resource is available on 20.04.2024

References

1. J. Aayush, H. Lin, A. Sahai, «Indistinguishability Obfuscation from Well-Founded Assumptions», https://eprint.iacr.org/2020/1003

Bulletin Of High Technology N 3 (31) 2024.-pp. 77-83.

2. C. Shangtong et al, «WASMixer: Binary Obfuscation for WebAssembly», https://doi.org/10.48550/arXiv.2308.03123
3. A. Durbet et al, «On the Leakage of Fuzzy Matchers», http://dx.doi.org/10.2139/ssrn.4592816
4. Z. Abideenet al, «An Overview of FPGA-inspired Obfuscation Techniques», https://doi.org/10.48550/arXiv.2305.15999
5. T. Zhao, H. Hu, C. Lu, «Unveiling the Role of Message Passing in Dual-Privacy Preservation on *GNN*», https://doi.org/10.48550/arXiv.2308.13513
6. *Н. Варновский, В. Захаров, Н. Кузюрин, А. Шокуров, «Современное состояние исследований в области обфускации программ: определения стойкости обфускации»,* https://doi.org/10.15514/ISPRAS-2014-26(3)-9
7. А. Мазин, Д. Щелкунов, «Исследование и разработка нового метода обфус-кации» // Вестник МГТУ им. Н.Э. Баумана, Серия «Приборостроение». 2009. №2. https://cyberleninka.ru/article/n/issledovanie-i-razrabotka-novogo-metoda-obfuskatsii
8. *J. Diaz, A. Garcia «Comparison of machine learning models applied on anony-mized data with different techniques»,* https://arxiv.org/abs/2305.07415
9. *S. K. Debray, «Compiler techniques for code compaction»,* https://doi.org/10.1145/349214.349233
10. *Р. Мулюков, А. Бородин, «Использование анализа недостижимого кода в статическом анализаторе для поиска ошибок в исходном коде программ». Труды Института системного программирования РАН, 2016, 28 (5), 145-158.*
11. Официальная страница загрузки вредоносного ПО, *Malware Bazaar Database. [Online]. Available, https://bazaar.abuse.ch/browse/, доступен* on 20.04.2024.
12. Официальная страница загрузки вредоносного ПО, Malwaredatabase. [Online].Available, http://vxvault.net/ViriList.php, *доступен* 20.04.2024.
13. Официальная страница сервиса проверки вредоносного ПО, https://www.virustotal.com/gui/home/upload доступен 20.04.2024
14. Официальная страница сервиса проверки вредоносного ПО, https://metadefender.opswat.com/ *доступен* 20.04.2024.
15. Исходный код программного обучения и полные результаты исследования. [Online]. Available: https://github.com/T-JN

ՄԵՔԵՆԱՅԱԿԱՆ ՈՒՍՈՒՑՄԱՆ ԿԻՐԱՌՄԱՄԲ ՎՆԱՍԱԲԵՐ ԾՐԱԳՐԱՅԻՆ ԱՊԱՀՈՎՄԱՆ ՕԲՖՈՒՍԿԱՑԻԱՅԻ ՄՈԴԵԼ

Թ.Վ. Ջամղարյան, Ա.Ա. Խեմչյան
Հայաստանի ազգային պոլիտեխնիկական համալսարան

Ներկայացված են ծրագրային ապահովման օբֆուսկացիայի համար գեներատիվ-մրցակցային ցանցի կիրառման ուսումնասիրության արդյունքները: Համեմատությունն իրականացվել է *Dotfuscator CE, Net Reactor, ProGuard* դետերմինիստիկ ծրագրային ապահովման հետ: *Athena, abc, cheeba, dyre, december_3, engrat, surtr, stasi, otario, dm, v-sign, tequila, flip, grum, mimikatz* ծրագրային ապահովման ելակետային կոդի փոփոխությունների ուսումնասիրությունն իրականացվել է *IDA Pro*-ի հետադարձ ճարտարագիտություն իրականացնող ծրագրային ապահովման միջոցով: Օբֆուսկացված վնասաբեր ծրագրերի հայտնաբերումն իրականացվել է համատեքստային անորոշ

Bulletin Of High Technology N 3 (31) 2024.-pp. 77-83.

մասնակի հեշավորման մեթոդի հիման վրա: Մշակված մեթոդի մոդելավորումն իրականացվել է Hyper-V վիրտուալ միջավայրում:

Բանալի բառեր. օբֆուսկացիա, հակադարձ ճարտարագիտություն, տվյալների հոսք, գեներատիվ-մրցակցային ցանց, մեքենայական ուսուցում, ssdeep, IDA Pro:

МОДЕЛЬ ОБФУСКАЦИИ ВРЕДОНОСНОГО ПО С ПРИМЕНЕНИЕМ МАШИННОГО ОБУЧЕНИЯ

Т.В. Джамгарян, А.А. Хемчян
Национальный политехнический университет Армении

Представлены результаты исследования применения генеративно-состязательной сети для обфускации программного обеспечения. Сравнение проводилось с детерминированными обфускаторами*DotfuscatorCE, NetReactor, ProGuard*. Исследование изменения исходного кода программного обеспечения *athena, abc, cheeba, dyre, december_3, engrat, surtr, stasi, otario, dm, v-sign, tequila, flip, grum, mimikatz*проводилось с помощью инструмента дизассмеблирования*IDAPro.* Обнаружение обфусцированного вредоносного ПО осуществлялось на основе метода кусочно-контекстного нечёткого хеширования.Исследование проводилось с целью изучения применения генеративно-состязательной сети в качестве обфускатора при заранее заданных наборах данных. Моделирование процессов проведено в виртуальной среде Hyper-V.

Ключевые слова: обфускация, реверс-инженеринг, поток данных, генеративно-состязательная сеть, машинное обучение, *ssdeep, IDA Pro.*

Submitted on 08.05.2024
Sent for review on 13.04.2024
Guaranteed for printing on 29.10.2024