Mathematical Problems of Computer Science 62, 72–81, 2024.

doi:10.51408/1963-0122

UDC 004.725:004.852

Obfuscated Malware Detection Model

Timur V. Jamgharyan, Vaghashak S. Iskandaryan and Artak A. Khemchyan

National Polytechnic University of Armenia, Yerevan, Armenia
e-mail: t.jamharyan@politechnic.am, Vagharshak.iskandaryan@gmail.com, a.khemchyan@politechnic.am

Abstract

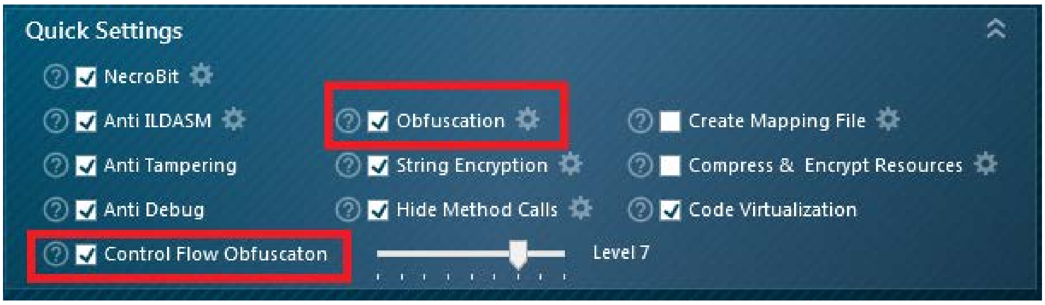
The paper presents the research results on the detection of obfuscated malware using a method based on *mean shift*. The research aimed to train neural networks included in the intrusion detection system to detect obfuscated malware. Detection of obfuscated malware using deterministic obfuscators is also discussed. Software solutions *Dotfuscator CE*, *Net Reactor*, and *Pro Guard* were used as deterministic obfuscators. *Athena, abc, cheeba, dyre, december_3, engrat, surtr, stasi, otario, dm, v-sign, tequila, flip, grum, mimikatz* were used as test malware. The results were verified using the *IDA Pro* tool and various intrusion detection systems. Process modeling was carried out in the Hyper-V virtual environment.

Keywords: Obfuscation, Reverse engineering, Data flow, Convolutional neural network, Machine learning, Clustering, *IDA Pro*, *Mean shift.*

Article info: Received 25 September 2024; sent for review 15 October 2024; accepted 18 November 2024.

1. Introduction

Training neural network models requires a large amount of training data, high-performance computing resources, and time. When training neural networks, it is important to evaluate the choice of machine and deep learning methods [1]. The use of neural networks for obfuscation of malware has created new requirements for the elements of the infrastructure protection system. In the case of obfuscated polymorphic and/or metamorphic malware, the situation is more complicated. Attackers who develop polymorphic and/or metamorphic malware obfuscate the zero version of the source code, then, due to the specific functioning of polymorphic and/or metamorphic malware, the source code changes, both with different replicas of this software and when the operating environment changes. All this is forcing network infrastructure (NI) security researchers to develop new methods to improve security. One of the problems that has not been completely solved is the detection of obfuscated malware. In particular, the [2] research defined the central task of the theory of obfuscation in relation to deterministic obfuscation means, and the researcch [3] formed a strict definition of «ideal» obfuscation. Reverse engineering of obfuscated malware leads to a certain version of this software, at best zero, but obfuscated, which allows you to hide the true signature of the malware. As indicated in [2], another unsolved problem is determining the potential of software that can be subjected to the obfuscation procedure. Solutions to obfuscate the source code of programs, before the advent of artificial neural networks, were mainly based on two fundamental algorithms: *Kohlberg’s* and *ChenxiWang’s algorithms* [4, 5]. Based on them, obfuscating malware has been developed (*Dotfuscator CE, Net Reactor, Pro Guard, COBF, HanzoInjection, Chimera,* etc.) that performs various types of obfuscation (lexical, preventive, data structures, data flow [6]). There are also obfuscators that are specific to a particular programming language [7]. But these obfuscators have one common property - determinism, which allows reverse engineer the protected software, if you have the appropriate hardware and/or time resources. Fig. 1 shows a part of the *Net Reactor* obfuscator menu, which allows you to protect software source code using various obfuscation methods.



Quick Settings

☑ NecroBit

☑ Anti ILDASM
☑ Obfuscation
□ Create Mapping File

☑ Anti Tampering
☑ String Encryption
□ Compress & Encrypt Resources

☑ Anti Debug
☑ Hide Method Calls
☑ Code Virtualization

☑ Control Flow Obfuscaton
Level 7

Fig. 1. Net Reactor obfuscator menu

The development of machine learning (ML) has taken obfuscation tools to a new level. Researchers are experimenting with different types and combinations of neural networks for software obfuscation [8, 10]. In [11], it is determined that «*computational resources for the operation of a neural network obfuscator are proportional to the number of trainable parameters*», that is, an increase in obfuscation parameters increases the consumed hardware resource when used as a tool for obfuscation of neural networks. The use of ML methods makes it possible to increase the degree of obfuscation of malware due to the introduced stochastic element. The algorithms used *(piecewise hashing, context-triggered piecewise hashing, statistically improbable features, block-based rebuilding)* [12, 14]), on the basis of which some malware detection tools are implemented, become ineffective when an attacker uses ML methods. Accordingly, an intrusion detection system (IDS) may miss this type of malware. When working with ML, an important task is to activate the necessary event handler (neural network) for a given type of malware. Inputting datasets into a detection neural network on which it is not trained increases the number of both type 1 and type 2 errors. Methods based on reverse engineering can partially solve this problem, but to do this, obfuscated malware must first be detected. Adding a stochastic element inherent to neural networks can «unmask» the source code of obfuscated malware by increasing the entropy value. IDS are capable of detecting obfuscated malware by analyzing the contents of the transmitted data packet. Attackers, trying to disguise malware in the network traffic flow, change the packet size of the transmitted data [15]. Accordingly, an urgent task is to detect malware obfuscated using neural networks in network traffic. Some researchers propose various methods and implementation tools to solve this problem [16, 18]. In this research, a clustering-based method using *mean shift*[1] is proposed to detect obfuscated malware. The choice of this method is based on the observation that when obfuscation by neural networks occurs, the entropy of the software source code increases, which can be detected by clustering code blocks and measuring the vector shift from the primary, «true» code. Solving this problem allows for more accurate calibration of IDS with ML in the mode of detecting obfuscated malware.

2. Formulation of the Problem

Research a model for using a *mean shift*-based method to detect obfuscated malware.

3. Proposed Solution

As a detector of obfuscated malware, it is proposed to use obfuscated malware trained on datasets of various dimensions: *athena, abc, cheeba, dyre, december_3, engrat, surtr, stasi, otario, dm, tequila, flip, grum, v-sign, mimikatz* convolutional neural network. The choice of a convolutional neural network is determined by the architecture of the network, which allows its variable restructuring. The *mean shift* method was used to detect changes in the code grouping value (constructing a clustering map by features) (1) [19]. Using *mean shift* as a research method enables the training of a convolutional neural network not only on the given datasets but also on additional ones within the shift vector. In this case, the shift vector is the number of changes to the malware source code obtained based on the clustering map.

$$m(x)=\frac{\sum_{x_i\in N(x)}K(x_i-x)x_i}{\sum_{x_i\in N(x)}K(x_i-x)}-x \qquad (1)$$

$m(x) - \textit{mean shift},$

$x - \textit{candidate centroid},$

$K(x)=1,\ x\to 0,\ \textit{otherwise}\ K(x)=0,$

$N(x)$ *as the neighborhood of samples within a given distance around x.*

Boundary conditions

- The size of the search vector is set manually (depending on the number of training datasets).
- K(x) = 0, recognition does not occur due to the combination of «false» and «true» source code of the analyzed software.
- The proposed solution is not scalable due to limitations of the algorithm itself.
- A change in the centroid by a value less than m(x) will not be detected (the source code itself changes and the software becomes inoperable).

4. Description of the Experiment

The Hyper-V role is installed in the Windows Server 2019 operating system environment. In the virtual environment installed IDS Snort, based on pfSense, IDS Suricata based on the

[1] Mean shift is a non-parametric feature-space mathematical analysis technique for locating the maxima of a density function, a so-called mode-seeking algorithm [19].

OPNsense and IDS ML based on Ubuntu v20.04 OS [20, 21]. All of them are united into a virtual local network. The convolutional neural network was trained on the basis of datasets obtained from the source code, software under study and sources [22, 27]. Training was carried out using the «unsupervised»[2] method. Parrot OS is also installed with the *Metasploit* framework installed to inject malware code into Windows Server 2019 with the *«test.local»* domain controller role installed, simulating the attacked server. Windows Server 2019 OS has 3 virtual network interfaces installed in different subnets with addresses (172.16.1.6/30, 172.16.2.6/30, 172.16.3.6/30). The networks of the second virtual adapter (Private 2) are displayed in a separate VLAN (Virtual Local Area Network, VLAN) using the hypervisor. All IDS are configured 1:1 NAT (Network Address Translation, NAT) for access from the external (Wide Area Network, WAN) interface to the domain controller. Using the *Metasploit* tool, obfuscated malware was injected into a domain controller deployed on Windows Server 2019. The virtual network diagram is shown in Fig. 2. The neural network was built from 20 neurons, with 37 weights for each. Initially, all the neural network weights were initialized to zero, to check the inputs/outputs and layers of the neural network and *mean shift*. The number of filters between the layers of the neural network, variable, depends on the *mean shift* step. The results were visualized using the *Tensor Board* library. The training datasets are converted to JSON (Java Script Object Notation, JSON) format. Traffic containing obfuscated malware first passed through various IDS before reaching the domain controller for comparison.

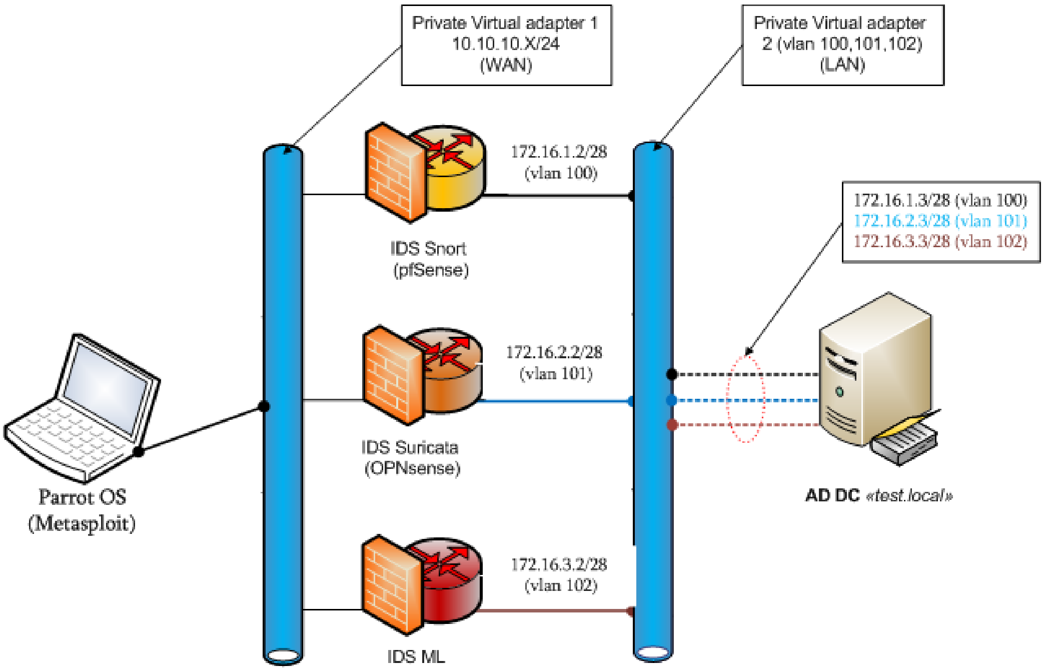
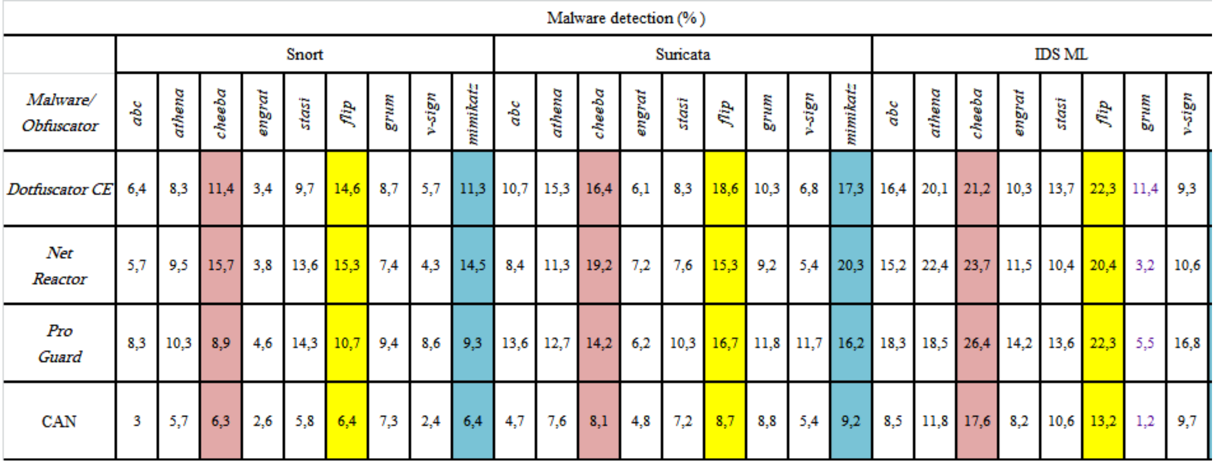


Fig. 2. Virtual network diagram

[2] 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.

Table 1. Obfuscated malware detection results



Malware detection (%)																									
	Snort									Suricata									IDS ML							
Malware/ Obfuscator	*abc*	*athena*	*cheeba*	*engrat*	*stasi*	*flip*	*grum*	*v-sign*	*mimikatz*	*abc*	*athena*	*cheeba*	*engrat*	*stasi*	*flip*	*grum*	*v-sign*	*mimikatz*	*abc*	*athena*	*cheeba*	*engrat*	*stasi*	*flip*	*grum*	*v-sign*
Dotfuscator CE	6,4	8,3	11,4	3,4	9,7	14,6	8,7	5,7	11,3	10,7	15,3	16,4	6,1	8,3	18,6	10,3	6,8	17,3	16,4	20,1	21,2	10,3	13,7	22,3	11,4	9,3
Net Reactor	5,7	9,5	15,7	3,8	13,6	15,3	7,4	4,3	14,5	8,4	11,3	19,2	7,2	7,6	15,3	9,2	5,4	20,3	15,2	22,4	23,7	11,5	10,4	20,4	3,2	10,6
Pro Guard	8,3	10,3	8,9	4,6	14,3	10,7	9,4	8,6	9,3	13,6	12,7	14,2	6,2	10,3	16,7	11,8	11,7	16,2	18,3	18,5	26,4	14,2	13,6	22,3	5,5	16,8
CAN	3	5,7	6,3	2,6	5,8	6,4	7,3	2,4	6,4	4,7	7,6	8,1	4,8	7,2	8,7	8,8	5,4	9,2	8,5	11,8	17,6	8,2	10,6	13,2	1,2	9,7

Figures 3,4,6, and 7 show visualized results of detecting obfuscated software *athena*, *dyre*, *surtr*, *grum*, *mimikatz*. Fig. 5 shows the reverse-engineered *mimikatz* malware obfuscated using a neural network.

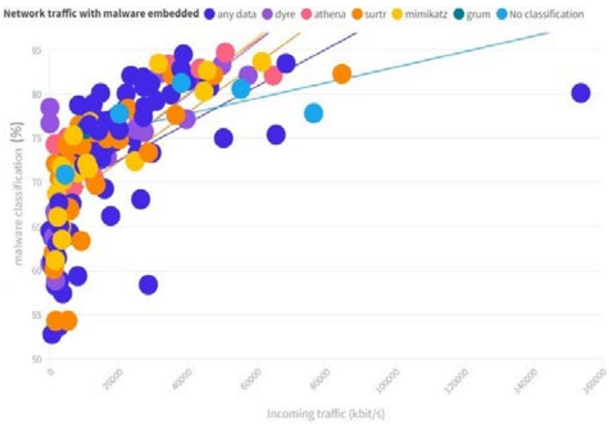


Fig. 3. Results of detection of obfuscated malware *athena, dyre, surtr, grum, mimikatz.*
I learning epoch

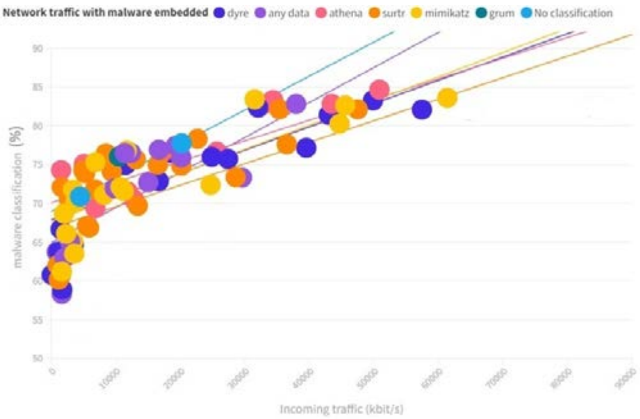
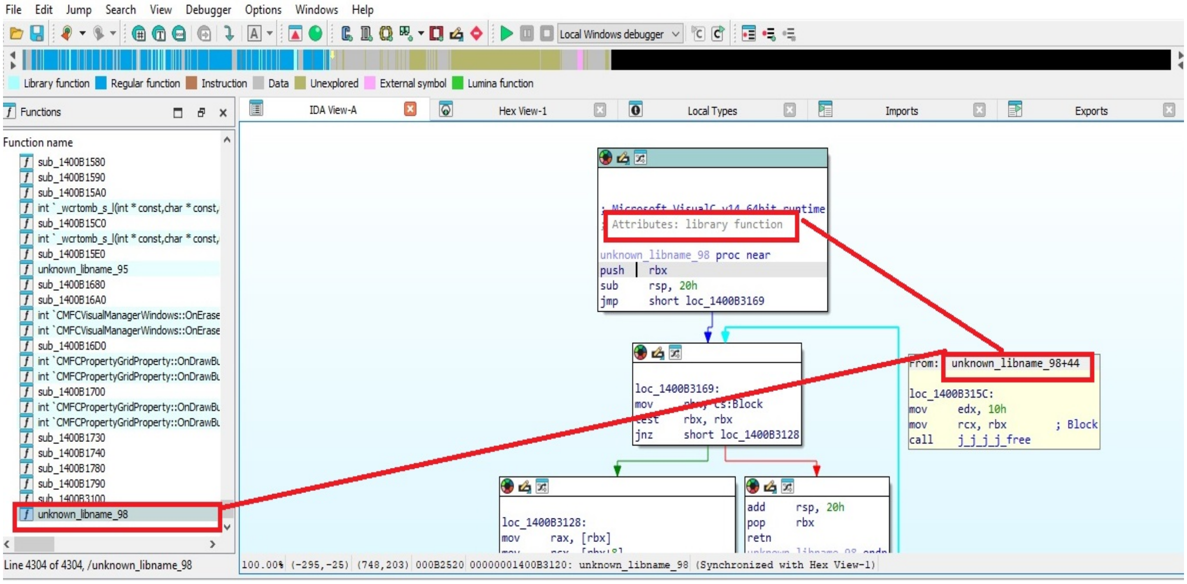


Fig. 4. Results of detection of obfuscated malware *athena*, *dyre*, *surtr*, *grum*, *mimikatz*.
III learning epoch

As can be seen from Fig. 5, the neural network adds libraries to the source code, which, without having an algorithmic effect on the execution of the code, create additional blocks of data, which change the software signature. Knowing the range of change for a given set of data it is possible to carry out a procedure for clustering their values. It is possible to predict which part of the code will be obfuscated at the next iteration (learning epoch) only probabilistically, which is consistent with the machine learning paradigm.

One of the differences between neural networks and deterministic obfuscators in the context of software obfuscation is that when obfuscating by neural networks, the source code of the obfuscated malware changes both at each iteration and at each training epoch. During an iteration, the values of the generated code variables change, and with a new training epoch, a new code is generated.



```
Function name
sub_1400B1580
sub_1400B1590
sub_1400B15A0
int `_wcrtomb_s_l(int * const,char * const,
sub_1400B15C0
int `_wcrtomb_s_l(int * const,char * const,
sub_1400B15E0
unknown_libname_95
sub_1400B1680
sub_1400B16A0
int `CMFCVisualManagerWindows::OnErase
int `CMFCVisualManagerWindows::OnErase
sub_1400B16D0
int `CMFCPropertyGridProperty::OnDrawBu
int `CMFCPropertyGridProperty::OnDrawBu
sub_1400B1700
int `CMFCPropertyGridProperty::OnDrawBu
int `CMFCPropertyGridProperty::OnDrawBu
sub_1400B1730
sub_1400B1740
sub_1400B1780
sub_1400B1790
sub_1400B3100
unknown_libname_98
Line 4304 of 4304, /unknown_libname_98
```

```
; Microsoft VisualC v14 64bit runtime
; Attributes: library function

unknown_libname_98 proc near
push    rbx
sub     rsp, 20h
jmp     short loc_1400B3169
```

```
loc_1400B3169:
mov     rbx, cs:Block
test    rbx, rbx
jnz     short loc_1400B3128
```

```
From: unknown_libname_98+44

loc_1400B315C:
mov     edx, 10h
mov     rcx, rbx        ; Block
call    j_j_j_j_free
```

```
loc_1400B3128:
mov     rax, [rbx]
```

```
add     rsp, 20h
pop     rbx
retn
```

```
100.00% (-295,-25) (748,203) 000B2520 00000001400B3120: unknown_libname_98 (Synchronized with Hex View-1)
```

Fig. 5. Reverse engineering *mimikatz* malware using the IDA Pro tool

The use of the *mean shift* method allows you to increase the detection of obfuscated malware by an average of (7÷9) % when it is obfuscated with deterministic obfuscators, and by (3÷5) % when obfuscated using neural networks (Table 1, Figures 6 and 7). In all cases, a training dataset is required, with a training sample value of at least 12% of the source code. During the learning process, when constant numerical values are received at the input of the neural network within the current iteration, the space of possible output values of the neural network is narrowed to the maximum value of the *mean shift* step. The best detection rate ((3÷5) %) was obtained by variable restructuring of the numerical value of the neural network activators when all weights were initialized with different random values. Constructing a clustering map of obfuscated malware using the *mean shift* method requires significant computing resources (the convolutional neural network used in the experiment was trained for 43 hours on equipment with a hardware configuration of RAM - 128 Gb, CPU - Intel Xeon E5-2694).

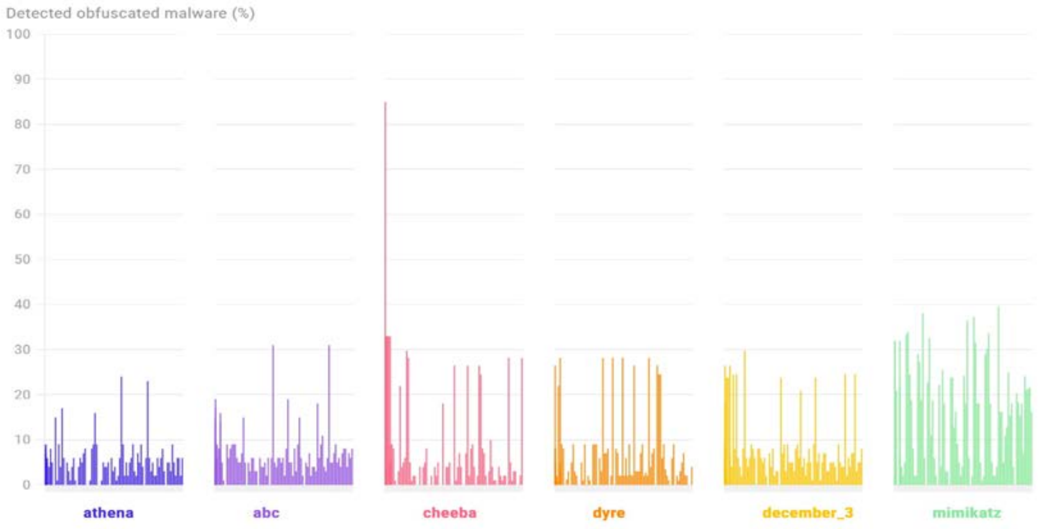


Fig. 6. Results of detection of obfuscated malware *athena*, *abc*, *cheeba*, *dyre*, *december_3*, *mimikatz* using *Suricata* IDS

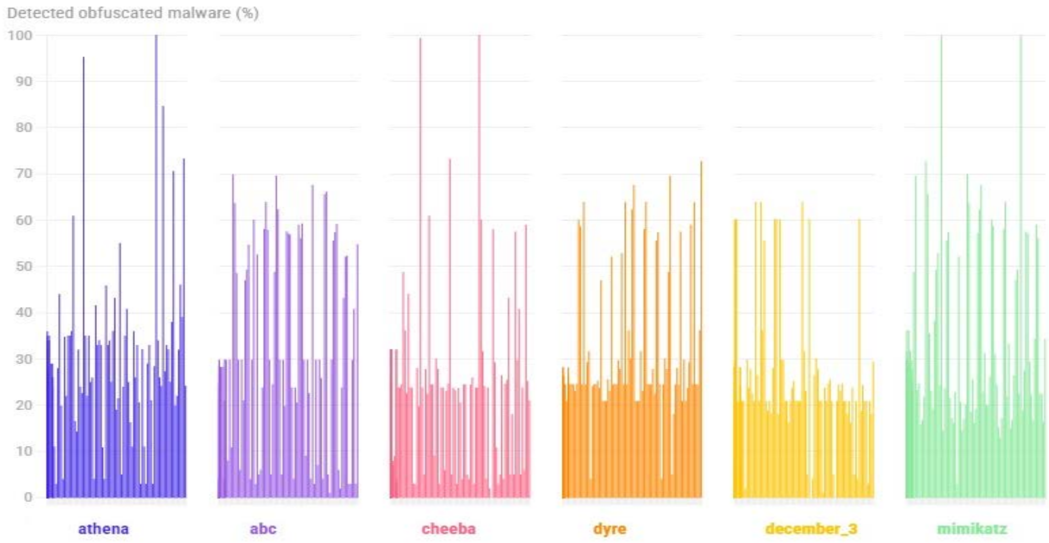


Fig. 7. Results of detection of obfuscated malware *athena, abc, cheeba, dyre, december_3, mimikatz* using IDS with ML (*mean shift* method)

The use of the *mean shift* method allows you to increase the detection of obfuscated malware by an average of (7÷9) % when it is obfuscated with deterministic obfuscators, and by (3÷5) % when obfuscated using neural networks (Table 1, Figures 6 and 7). In all cases, a training dataset is required, with a training sample value of at least 12% of the source code. During the learning process, when constant numerical values are received at the input of the neural network within the current iteration, the space of possible output values of the neural network is narrowed to the maximum value of the *mean shift* step. The best detection rate ((3÷5) %) was obtained by variable restructuring of the numerical value of the neural network activators when all weights were initialized with different random values. Constructing a clustering map of obfuscated malware using the *mean shift* method requires significant computing resources (the convolutional neural network used in the experiment was trained for 43 hours on equipment with a hardware configuration of RAM - 128 Gb, CPU - Intel Xeon E5-2694).

5. Conclusion

This paper studies the use of the *mean shift* method for detecting obfuscated malware using a neural (generative-adversarial) network. The results obtained make it possible to more accurately calibrate the IDS with ML, as well as to construct a map of clustering features, which creates an additional learning model for neural networks from the IDS with ML. Compared to similar studies, the use of the *mean shift* method is justified if there are appropriate training datasets that correlate with the «true source code». As the training epoch increases, the number of type 2 errors also increases leading to overfitting. The use of the *mean shift* method is to search for obfuscated polymorphic malware is unjustified, since the algorithm converges at a small value of centroids, which is important for polymorphic malware. The proposed method makes it possible to increase the reliability of the operation of IDS with ML while simultaneously activating several neural networks in the mode of detecting obfuscated malware. Based on the results obtained, improvements were made to the IDS with ML and other elements of the network infrastructure according to the specified parameters [29]. All research results are presented in [30].

References

[1] Microsoft official website https://learn.microsoft.com/ru-ru/azure/machine-learning/concept-deep-learning-vs-machine-learning?view=azureml-api-2

[2] Yu. Livshits, laboratory of mathematical logic at PDMI, "Obfuscation of programs", 2004. https://logic.pdmi.ras.ru/~yura/of/survey1.pdf

[3] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan and K. Yang, "On the (im) possibility of obfuscating programs", *Advances in Criptology Crypto 2001*, LNCS 2139, Springer-Verlag, pp. 1-18, 2001.

[4] C. Collberg and Jasvir Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*, Addison-WesleyProfessional, 2009.

[5] C. Wang, J. Hill, J. Knight and J. Davidson, "Software tamper resistance: obstructing static analysis of programs", Technical Report. University of Virginia, Charlottes ville, VA, USA., 18 p., 2000.

[6] K. Monappa, *Learning Malware Analysis*, Packt, Birmingem-Mumbai, 2019.

[7] Official website of the Java programming language obfuscator, Zelix. https://www.zelix.com/

[8] M. S. Karvandi et al., "The reversing machine: reconstructing memory assumptions", https://doi.org/10.48550/arXiv.2405.00298

[9] C. Patsakis, F. Casino and N. Lykousas, "Assessing LLMs in malicious code deobfuscation of real-world malware campaigns", https://doi.org/10.48550/arXiv.2404.19715

[10] S. Hasan and A. Dhakal, "Obfuscated malware detection: investigating real-world scenarios through memory analysis", https://doi.org/10.48550/arXiv.2404.02372

[11] V. Eliseev, "Artificial neural networks as a mechanism for obfuscation of calculations", https://doi.org/10.17223/2226308X/12/46

[12] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing", *Digital Investigation*, vol. 3, Supplement, pp. 91-97, 2006, doi: 10.1016/j.diin.2006.06.015

[13] L. Chen and G. Wang, "An efficient piecewise hashing method for computer forensics", *First International Workshop on Knowledge Discovery and Data Mining (WKDD 2008)*, Adelaide, SA, Australia, pp. 635-638, 2008, doi: 10.1109/WKDD.2008.80

[14] V. Roussev, "Building a better similarity trap with statistically improbable features" *2009 42nd Hawaii International Conference on System Sciences*, Waikoloa, HI, USA, pp. 1-10, 2009, doi:10.1109/HICSS.2009.97

[15] M. Alyami, A. Alghamdi, M. Alkhowaiter, C. Zou and Y. Solihin, "Random segmentation: new traffic obfuscation against packet-size-based side-channel attacks", https://doi.org/10.48550/arXiv.2309.05941

[16] I. Nunes, S. Hwang, S. Jakkamsetti, G. Tsudik, "Privacy-from-Birth: protecting sensed data from malicious sensors with VERSA", https://doi.org/10.48550/arXiv.2205.02963

[17] M. Rosen, J. Parker and A. Malozemoff, "Balboa: bobbing and weaving around network censorship" https://doi.org/10.48550/arXiv.2104.05871

[18] L.Wang, H. Kim, P. Mittal and J. Rexford, "Programmable in-network obfuscation of traffic", https://doi.org/10.48550/arXiv.2006.00097

[19] Y. Cheng, "Mean shift, mode seeking, and clustering", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790-799, Aug. 1995, doi: 10.1109/34.400568

[20] T. V. Jamgharyan, "Research of obfuscated malware with a capsule neural network", *Mathematical Problems of Computer Science*, vol. 58, pp. 67–83, 2022. doi: 10.51408/1963-0094

[21] T. V. Jamgharyan, “Modernization of Intrusion Detection System using Generative Model”, *Defense-Academic Journal, National Defense Research University, Haykakan Banak (Armenian Army), vol.* 2(108), pp. 69-79, 2021, https://razmavaraget.files.wordpress.com/2022/01/hb2-final.pdf

[22] Malware Bazaar Database. [Online]. Available https://bazaar.abuse.ch/browse/

[23] Malware database. [Online]. Available http://vxvault.net/ViriList.php

[24] A free malware repository for researches. [Online]. Available https://malshare.com/

[25] Malware repository. [Online]. Available https://avcaesar.malware.lu/

[26] Malware repository. [Online]. Available https://www.virusign.com/

[27] Viruses repository. [Online]. Available https://virusshare.com/

[28] T. V. Jamgharyan and A. A. Khemchyan, “Malware obfuscation model using machine learning”, *Bulletin of High Technology,* vol. 3, no. 31, pp. 77-83, 2024. doi: 10.56243/18294898-2024.3-77

[29] T. V. Jamgharyan, T. N. Shahnazaryan, “A studu of a model of neural network application in the decoy infrastructure in the defence sphere”, *Defence-Academic journal, National Defence Reseach Univers*ity, Haykakan Banak (*Armenian Army*), 2(112), pp. 71-83, 2024. DOI: 10.61760/18290108-ehp24.2-71

[30] All research results available on https://github.com/T-JN

Օբֆուսկացված վնասաբեր ծրագրային ապահովման հայտնաբերման մոդել

Թիմուր Վ. Ջամղարյան, Վաղարշակ Ս. Իսկանդարյան և Արտակ Ա. Խեմչյան

Հայաստանի Ազգային Պոլիտեխնիկական Համալսարան, Երևան, Հայաստան
e-mail: t.jamharyan@politechnic.am, Vagharshak.iskandaryan@gmail.com, a.khemchyan@politechnic.am

Ամփոփում

Հոդվածում ներկայացված են օբֆուսկացված վնասաբեր ծրագրային ապահովման հայտնաբերման հետազոտության արդյունքները՝ օգտագործելով *mean shift* մեթոդը։ Հետազոտությունն իրականացվել է ներխուժման հայտնաբերման համակարգում ընդգրկված նեյրոնային ցանցերի ուսուցման նպատակով։ Դիտարկվել է դետերմինիստական օբֆուսկատորներով ու նեյրոնային ցանցերի կիրառմամբ օբֆուսկացված վնասաբեր ծրագրային ապահովման հայտնաբերումը։ Որպես թեստային վնասաբեր ծրագրային ապահովում օգտագործվել է՝ *athena, abc, cheeba, dyre, december_3, engrat, surtr, stasi, otario, dm, v-sign, tequila, flip, grum, mimikatz-ը։* Արդյունքների վերիֆիկացիան իրականացվել է *IDA Pro* գործիքի և տարբեր ներխուժման հայտնաբերման համակարգերի միջոցով։ Պրոցեսների մոդելավորումը իրականացվել է Hyper-V վիրտուալ միջավայրում։

Բանալի բառեր՝ օբֆուսկացիա, կլաստերիզացիա, ներխուժման հայտնաբերման համակարգ, ցանցային ենթակառուցվածք, *IDA Pro, mean shift.*

Модель Обнаружения Обфусцированного Вредоносного ПО

Тимур В. Джамгарян, Вагаршак С. Искандарян и Артак А. Хемчян

Национальный Политехнический Университет Армении,Ереван, Армения
e-mail: t.jamharyan@politechnic.am, Vagharshak.iskandaryan@gmail.com, a.khemchyan@politechnic.am

Аннотация

В статье представлены результаты исследования обнаружения обфусцированного вредоносного программного обеспечения с применением метода на основе *mean shift*. Исследование проводилось с целью обучения нейронных сетей, входящих в состав системы обнаружения вторжений, обнаружению обфусцированного вредоносного программного обеспечения. В качестве детерминированных обфускаторов использовались программные решения *Dotfuscator CE*, *Net Reactor*, *Pro Guard*. В качестве тестового вредоносного программного обеспечения использовалось *athena, abc, cheeba, dyre, december_3, engrat, surtr, stasi, otario, dm, v-sign, tequila, flip, grum, mimikatz,* различных версий. Верификация результатов проводилась с помощью инструмента *IDA Pro* и различных систем обнаружения вторжений. Моделирование процессов проведено в виртуальной среде Hyper-V.

Ключевые слова: обфускация, реверс-инженеринг, поток данных, сверточная нейронная сеть, машинное обучение, кластеризация, *IDA Pro, mean shift*.