Mathematical Problems of Computer Science 59, 69–81, 2023.
doi: 10.51408/1963-0103

UDC 004.725, 004.852

Research of Model Increasing Reliability Intrusion Detection Systems

Timur V. Jamgharyan

National Polytechnic University of Armenia, Yrevan, Armenia
e-mail: t.jamgharyan@yandex.ru

Abstract

The paper presents the results of the using, a recurrent neural network to detect malicious software as part of the *Snort* intrusion detection system.The *research* was conducted on datasets generated on the basis of *athena, dyre, engrat, grum, mimikatz, surtr* malware exploiting vulnerability *CVE-2022-20685* in the *Snort* intrusion detection system. Processing of input traffic data was carried out before the *frag-3* and *modbus* preprocessors. The method of *k nearest neighbors* was used as a mathematical apparatus. The simulation of the developed software at different iterations.

All research results are presented in https://github.com/T-JN

Keywords: Machine learning, Dataset, Malware, Preprocessor, Metasploit, k nearest neighbors method, Intrusion detection system.

Article info: Received 8 January 2023; send to review 7 February 2023; accepted 7 March 2023.

1. Introduction

The intrusion detection systems (IDS) include many different software components designed to detect various types of traffic with an embedded malicious component. Detection is carried out according to a set of rules that are configured based on the threat model and security policies. The security architecture of the Network Infrastructure (NI) is built taking into account possible attacks according to various models: triad CIA (Confindentiality, Integrity, Availability, CIA), Parker's hexad [1]. Network IDS, unlike host IDS, detect attacks directed at the network segment and contain a set of complementary rules and security scripts that can neutralize an attack on the network. Unlike host-based IDS, network-based IDS require more computing resources due to the fact that a larger set of rules and detectors is activated during their operation [2]. When using host IDS in the Infrastructure for a fleet of computing systems running Linux OS, can disable

the rules for Windows (or another OS), but hardly possible for a network IDS, since different operating systems are used in the Infrastructure. Modern IDS are able to detect various types of attacks at different levels of the OSI (Open System Interconnection, OSI) model: bad traffic, system scanning, the use of known exploits to attack over various protocols, various backdoors, various known malware [3]. A significant limitation of systems for analyzing network traffic and the state of NI is the algorithmic and functional determinism inherent in them.

An important issue of Infrastructure security is the reliability of the processed data of the IDS itself (*data reliability – is, the property of the processed data not to have hidden errors* [4]). The processing of data streams in the IDS itself is determined by the functioning algorithms, data presentation formats, and the formalization of signature classifiers. Protecting the IDS signature database (both remote and local) is also one of the most important tasks. If the signatures database has been attacked for availability, then when a new vulnerability appears, the IDS will not receive the necessary signature and the Infrastructure perimeter will become vulnerable [5]. The development of M2M (Machine-to-Machine, M2M) and ML (Machine learning, ML) technologies has increased the capabilities of both attack and defense tools. Various researchers are conducting research on increasing (improving) various parameters of IDS with ML [6, 7, 8]. One of the parameters that improves when using ML modules as part of a standard IDS is its variability. Unlike deterministic IDS, IDS with ML are capable of forming a multi-criteria sample on the basis of which the detector operation scheme is formed within the given constraints. But IDS with ML have certain limitations when integrating them into the NI architecture. In particular, ML IDS are very sensitive to various implementations of *«noise attacks» («noise attack» is a variant of an availability attack in which a large number of random and meaningless fragmented packets are sent to the attacked system, some of which contain malware* [9]). A dangerous consequence of a «noise attack» on a ML network IDS is that attackers «attack» it for a long time with streams of datasets that cause false positives, «teach» the ML IDS discriminator to be immune to this type of traffic (creating a cyclic chain of operations: false positive--true negative--false negative--true positive, which overload both the IDS itself and the SIEM system (Security information and event management, SIEM).

Various manufacturers combine IDS modules into different classes, which allows you to quickly reconfigure the IDS itself for specific tasks. In particular, for Snort open source IDS, there are many different types of preprocessors (*frag-3, stream, performance monitor, SMTP, POP, IMAP, SSH, DNS, DCE/RPC, SIP preprocessors, reputation preprocessor, modbus preprocessor*) each of which is functionally is responsible for handling the given protocol and/or data type.

- *IDS preprocessor is a software module that receives data from the network traffic decoding module and outputs them to the input of intrusion detection modules.*

As stated in the article «Attacks on Machine Learning Systems» [10], the most vulnerable part of the ML IDS is the traditional IDS component (the deterministic part of the IDS). ML systems, like any other, will be hacked using vulnerabilities in these traditional components. The use of ML at the preprocessor level is due to the fact that when developing an IDS with ML, it is not enough to create a functioning model that can detect a threat not described in a set of rules (signatures) or generate new ones based on «known» signatures, but it is also necessary to protect the IDS itself from probable infection with malware that can compromise the reliability of the results issued by IDS.The choice of using a neural network at the preprocessor level is also due to the fact that the IDS, which has a neural network in its component composition after the preprocessor, is able to protect the NI, since malware not detected by standard datasets (described in the signature/rule database) will be detected with varying probability neural network. But with a «noise attack», the target is the IDS itself, which, when taken out of the reliable functioning mode, will no longer detect malware. Undescribed at the preprocessor level,

malicious data embedded in IDS can be detected using performance preprocessors that evaluate various kinds of statistics. But the problem is that, having determined the type of network IDS, attackers can design an attack taking into account the work of preprocessors, and malware embedded in the IDS itself will not go beyond the allowable statistical deviations. A lot of research has been devoted to the task of applying machine learning as part of IDS, but only a small part of them explores the use of machine learning at the preprocessor level. This limitation, in particular, is due to the fact that the «response» of the neural network is probabilistic in nature and it is necessary to introduce clear boundaries for the neural network itself. Otherwise, the neural network will be an event generator, which will be classified as an attack by the IDS detection modules. Thus, there is a recursion to the problem of stability and integrity of both the IDS and the NI as a whole [11]. *This research explores the potential of a recurrent neural network (RNN) to detect malware at the preprocessor level*. The choice in the research of RNN from the entire set of neural networks is determined by the fact that RNN form a directed sequence between elements, which allows processing a series of events in time (this characteristic allows granular processing of fragmented datasets). The relevance of the work lies in the ever-increasing role of IDS with ML in the NI security architecture and the increasing security requirements of the IDS itself. The use of a neural network at the preprocessor level will increase the reliability of malware detection results without affecting the main IDS signature database, which will reduce the attack surface for the IDS itself. The novelty of the research lies in the application of the k nearest neighbors (k Nearest Neighbors, kNN) method to detect malware in IDS *before preprocessors*.

- *The k nearest neighbors method is a metric algorithm for classifying objects.*

Malicious software athena, dyre, engrat, grum, mimikatz, surtr obtained from publicly available sources was used as calibration data [12--15]. The choice of the kNN method is determined by the fact that it is necessary to minimize the value of the preprocessor error, and for this it is necessary to carry out a preliminary grouping and classification of unknown input datasets in normalized traffic.

- *Traffic normalization - modification of packets of protocols of the transport, and network levels for their subsequent processing by IDS detection modules.*

2. Formulation and Description the Problem

It is necessary to detect a malicious dataset in normalized traffic.

The mathematical model construction was carried out on the basis of the formulas obtained in the sources [16,17]. There are network traffic X inputs that contain malware fragments (1).

$$X^m = \{(x_1, y_1), \dots, (x_m, y_m)\}, \quad (1)$$

where,

x_m- network traffic datasets that do not contain malicious components,

y_m- network traffic datasets containing malicious components,

m- number of the analyzed packet of the input dataset.

On the set of input traffic data sets, the distance function $x\rho(y, y')$ is given. The greater the value of the distance function, the less similar the entities are y, y', where y'- the minimum size of a malware dataset that can be uniquely identified and classified with respect to y. For any entity v in the data package, arrange the objects x_i in ascending order (2).

$$\rho(v, x_{1;v}) \le \rho(v, x_{2;v}) \le \dots \le \rho(v, x_{m;v}), \quad (2)$$

where $x_{i;v}$ the set of network traffic data that is the i-th neighbor of the entity v. Similarly for the i -th neighbor of the entity v in the dataset $y_{i;v}$. Using the formula (3 from the source [17], we determine the malicious kNN components for the traffic arriving in the NI.

$$\alpha(v) = \arg\max_{y \in Y} \sum_{i=1}^{m} [y(x_{i;v}) = y]\, \omega(i, v), \tag{3}$$

where, $\omega(i, v)$- a given weight function that evaluates the degree of importance of the i-th neighbor for the classification of the entity v. By changing the $\omega(i, v)$ value, you can get different versions of the k nearest neighbors method (4).

$$\omega(i, v) = [i \leq k]. \tag{4}$$

When $\omega(i, v) = [i = 1]$ malware is detected only in the given single value ω. That is, the RNN is only able to detect the malware datasets it was trained on. A graphical representation of a RNN is shown in Fig. 1.

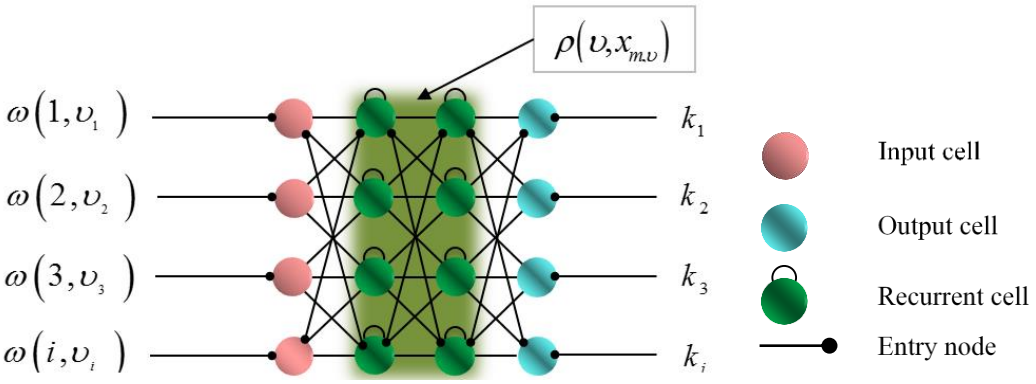

Fig. 1. Recurrent neural network.

Attackers can load malware into the IDS itself not in a single package, but in fragments (using the built-in *frag-3* preprocessor as an internal attack tool), then the research task of grouping and classifying malware fragments arises. Standard IDS do not cope with this task very effectively, but ML IDS, in the presence of a training set, are able to solve this problem. The disadvantage of ML IDS is that they can produce unreliable results if the preprocessor responsible for a particular type of traffic/protocol is «damaged» as a result of a *«noise attack»*. A particular danger lies in the fact that any traffic entering the IDS preprocessors (both ML and deterministic) is not checked for malicious components, since the task of the preprocessor is to «reformat» traffic for processing by detectors.

3. Task Statement

It is necessary to develop and programmatically implement an algorithm and, based on it, software that integrates a RNN capable of solving the problem of grouping and classification with the IDS preprocessor.

4. Boundary Conditions

1. The smallest fragment of the malware file (ξ) that can be classified $\xi = 20byte$ (detection was carried out using context-piecewise hashing (Context Triggered Piecewise Hashing, CTPH), which is discussed in detail in [18].
2. The delay in the processed module should not cause a «signal race». Traffic from the output of the preprocessor module to the input of the detection modules must be sent synchronously. As part of this condition, an additional restriction has been introduced - only UDP (User Datagram Protocol, UDP) traffic is processed.
3. The hardware must support the parallel computing mode.

The developed software connects the RNN to *frag-3* and *modbus* preprocessors (*frag- 3* preprocessor for defragmenting an IP packet, *modbus* - preprocessor for processing data from a variety of devices operating in SCADA networks (Supervisory Control And Data Acquisition, SCADA).Since the *frag-3* preprocessor is designed to build packages, using a trained RNN can neutralize the process of «assembling» malicious packages inside the IDS, increasing the level of reliability of its functioning. On Fig.2 shows a diagram of the *Snort* IDS with the proposed data processing software implemented on RNN.

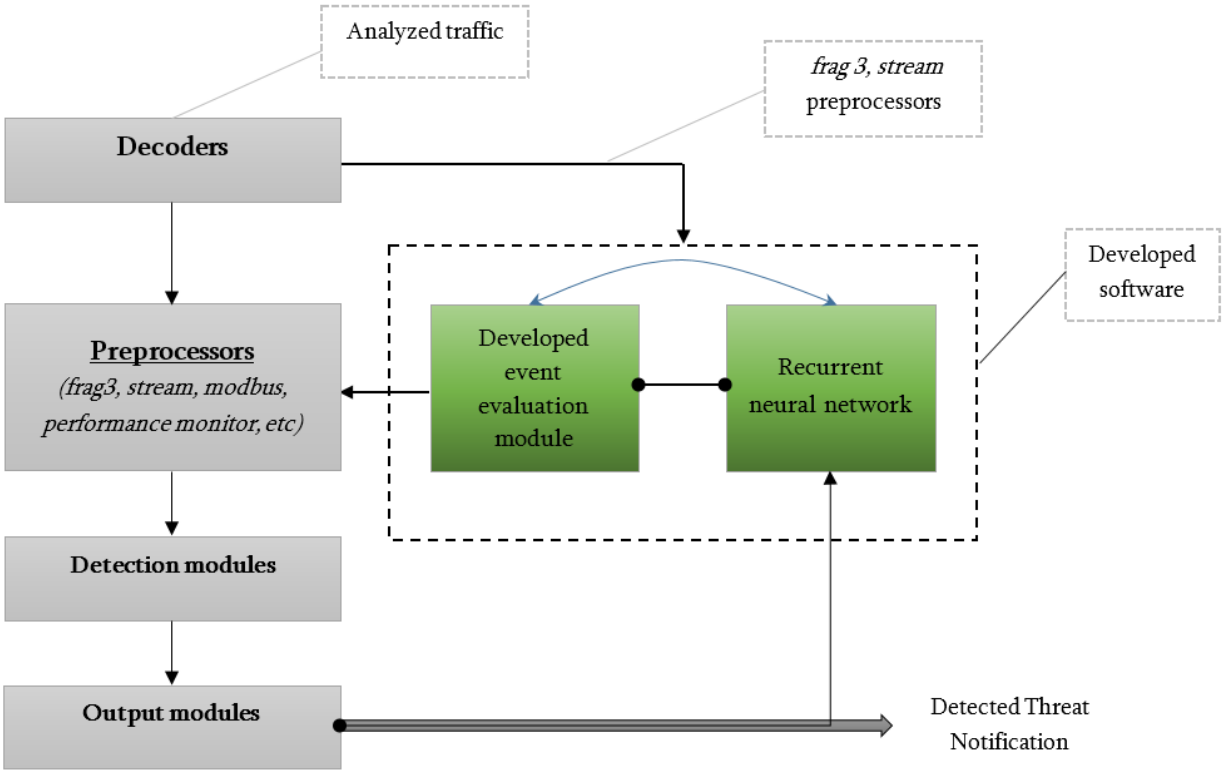

Fig. 2. Snort IDS with developed data processing software.

5. Description of the Module

The network traffic coming from the decoders is directed to the preprocessor processing module (standard operation of the *Snort* IDS). The traffic that should processed by the *frag - 3* and *modbus* preprocessors is sent to the developed module based on the RNN. After processing according to the developed algorithm, this traffic is again sent to the standard detection modules. The task of the module is to carry out the primary «cut-off» of possible malware and protect the IDS itself from being modified by malware.

The developed algorithm is shown in Fig. 3.

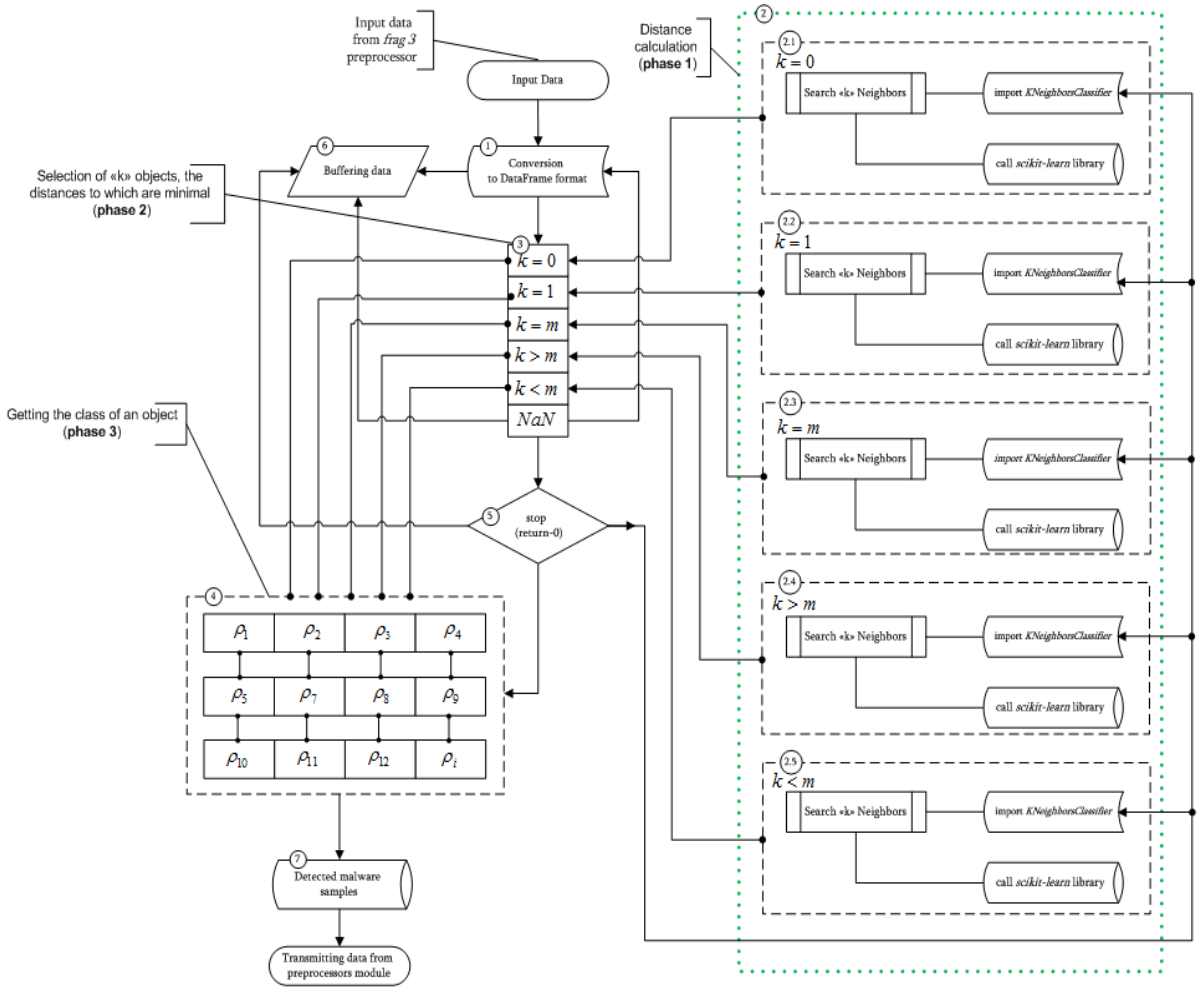

Fig. 3. Developed algorithm.

Algorithm operation

The software that searches for fragmented malware receives network traffic datasets from a decoder (*Snort* IDS a low-level interceptor) as input. Only traffic that must be processed by the *frag-3* and *modbus* preprocessors is subject to processing.

Step 1. Converting received datasets to «Data Frame». This conversion is necessary to speed up the work of the RNN, since the traffic not processed by the developed module goes directly to

the preprocessor module and the processing delay should not exceed the boundary conditions (boundary condition 2).

Step 2 phase 1. *Calculation of the distance from the target object, which must be classified to each of the sample objects (traffic).* Computing a distance metric between likely malware datasets. All calculations are performed in parallel mode (boundary condition 3),

- **2.1 k=0** calculation of the distance metric and detection of malicious datasets is not performed, since the classification of malicious and non-malicious datasets is impossible,
- **2.2 k=1** the distance between malicious and non-malicious datasets is constant (k=const). Only those malicious datasets that fall within the specified distance metric are detected,
- **2.3 k=m** continuous detection mode.Upper limit: the value of m that the hardware can handle,
- **2.4 k>m** malicious datasets are not detected,
- **2.5 k<m** malicious datasets are detected down to the minimum CTPH value. All calculations were based on the scikit-learn ML library (using instances of the *kNeighborsClassifier* class).

Step 3 phase 2. *Selection of k objects from the sample, the distances to which are minimal.*

The RNN to fed only datasets, where corresponding to paragraphs 2.2, 2.3, 2.5. When a number value with an undefined result NaN (Not-a-Number, NaN) appears in the handler, the execution of the entire program is «stopped», which resets all values to zero (step 5).

Step 4 phase 3. *Obtaining a class of sample objects based on the most frequently occurring k.* Setting the «weights» of the RNN. The weight setting is determined by the number of malware hash values detected by the CTPH method. Increasing the value $\rho(\upsilon_i, x_{m;\upsilon})$ (increasing the number of hits) for a certain type of dataset increases the «weight» of this dataset in the RNN. The output is a class of malware datasets.

Step 5. Stop and reset all values when NaN values appear in the dataset.

Step 6. Buffering values one step before zeroing. The buffer always contains n-1 dataset values (the n-dataset currently being processed).

Step 7. Detected malware datasets.

Step 8. Transfer of traffic to the input of the preprocessor module.

All class instances are implemented based on the *StandardScaler* library. The training was carried out on the basis of the *fit* software library.

6. Description of the Experiment

In Windows Server 2016 Standard operating system environment installed the Hyper-V role (Based on the Dell Power Edge T-330 server). A software-defined network (SDN) has been deploy, in which Parrot OS is installed with the Metasploit framework and Ubuntu v20.04 OS in which are installed: IDS *Snort* version 2.9.18, *Clion* development environment and developed software. The introduction of traffic with malware that could lead to a denial of service for the *Snort* IDS and an attack on the Infrastructure was carried out using the Metasploit framework based on the Parrot OS pentest distribution kit. The malicious input was based on a *pcap* network traffic dump file. The choice of version 2.9.18.1 of the *Snort* IDS is due to the fact that in this version there is a vulnerability *CVE-2022-20685* (*CVE-2022-20685 Snort* IDS vulnerability leading to a denial of service, bypassing security restrictions and compromising the system[19]) when exploited, attackers can inject malware into the IDS itself and attack the Infrastructure. With the correct operation of the developed software, the attack should be detected, which will make it possible to further check the effectiveness of the software for possible and probable

unknown attacks. Through this vulnerability, *athena, dyre, engrat, grum, mimikatz, surtr* malware was introduced into the virtual Infrastructure. The Windows Server 2016 operating system, which is the *test.local* domain controller, and the Windows 10 client machine were used as the protected Infrastructure. To increase the reliability of the experiment results, all virtual machines are connected to each other by a *private* virtual adapter and connected to different VLAN (Virtual Local Area Network, VLAN, with vlan ID=100 and vlan ID=101). Network address translation (NAT) is configured between virtual networks 172.16.0.0/30 and 192.168.0.0/29.

The experiment was carried out in 2 stages.

Stage 1.

Injection of *mimikatz* malware through *CVE-2022-20685* with *kNN*-based detection software disabled. In the first case, the IDS did not detect the intrusion, and the *mimikatz* software implemented through the Snort IDS in the «*noise attack*» mode compromised the domain administrator's password and did not register the *Snort* network IDS in any way.

Stage 2.

Introduction of various types of malware (*athena, dyre, engrat, grum, mimikatz, surtr*) into the Infrastructure through a vulnerability in the *Snort* network IDS. The *mimikatz, surtr, engrat, and grum* malware were detected immediately, while the athena and dyre malware was detected after the second iteration.

The scheme of the experiment is shown in Fig. 4.

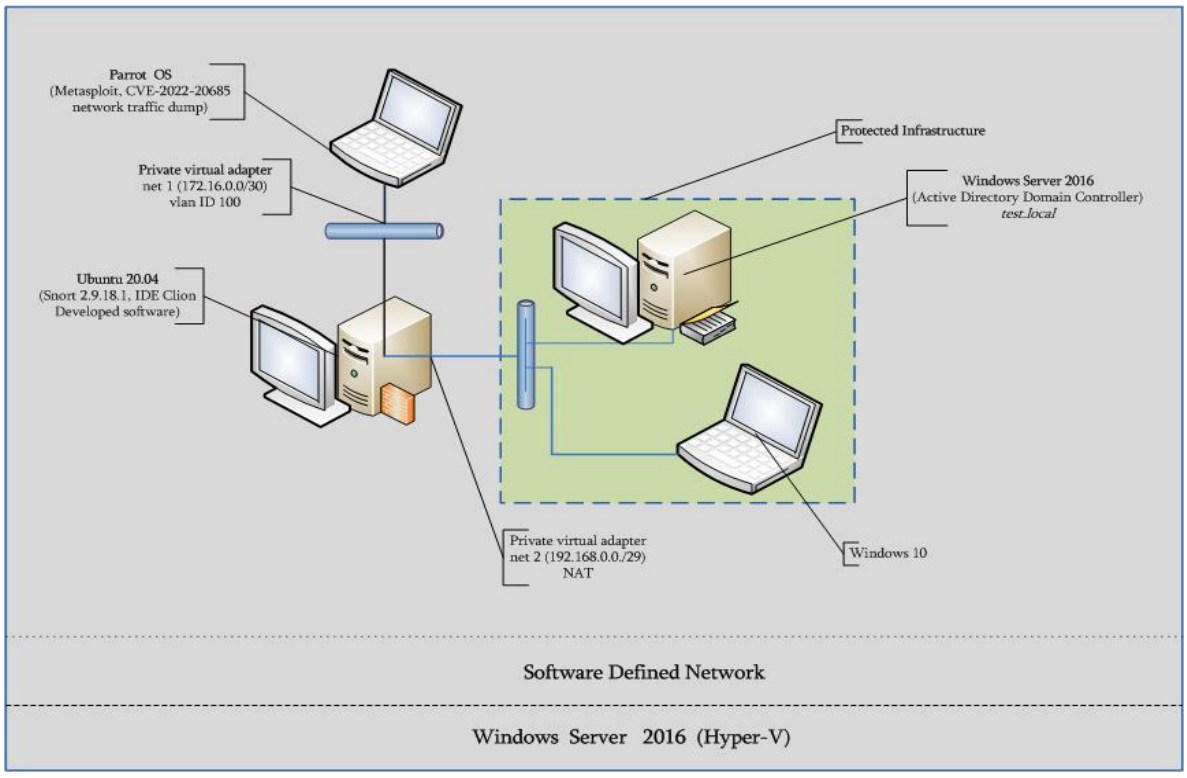

Fig. 4. Scheme of the experiment in SDN.

7. Results

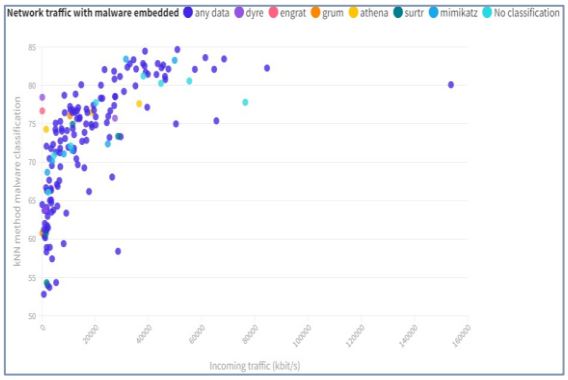

Fig. 5. Visualization of datasets classified by the kNN method of malware (I-iteration)

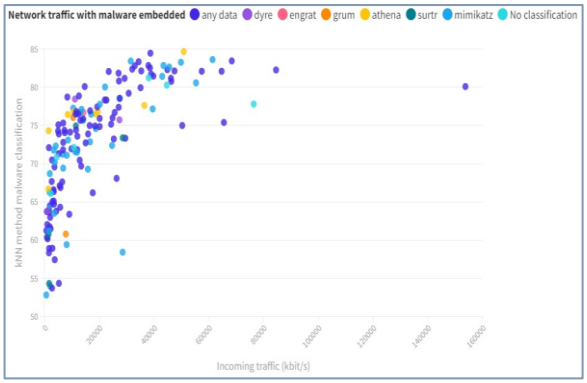

Fig. 6. Visualization of datasets classified by the kNN method of malware (II-iteration)

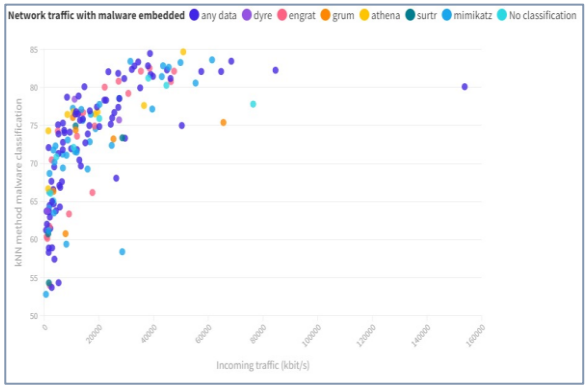

Fig. 7. Visualization of datasets classified by the kNN method of malware (III-iteration)

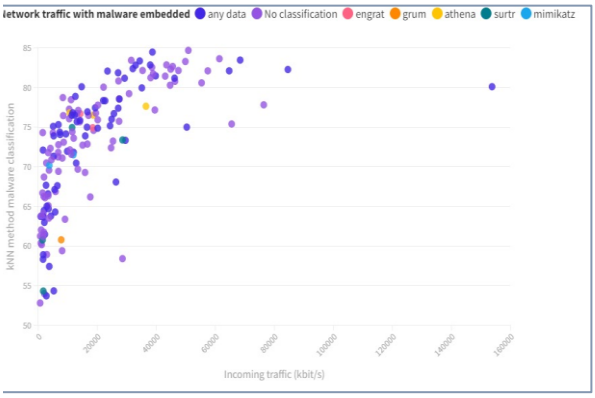

Fig. 8. Visualization of datasets classified by the kNN method of malware (IV-iteration)

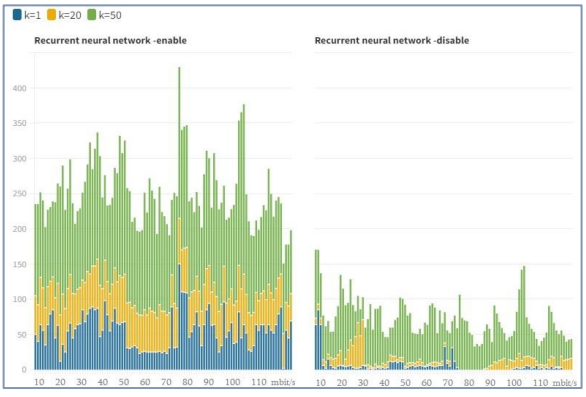

Fig. 9. Visualization of datasets classified by the kNN method of malware. k=1, 20, 50.

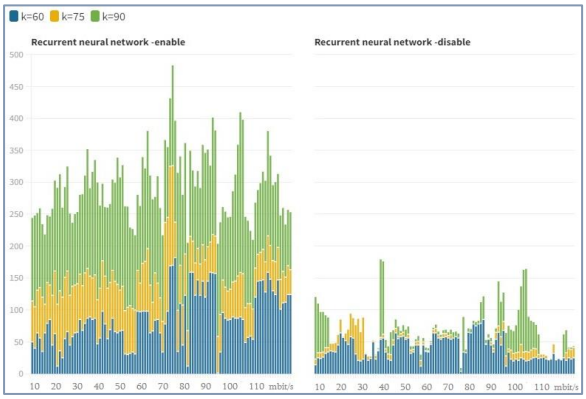

Fig. 10. Visualization of datasets classified by the kNN method of malware. k=60, 75, 90.

As part of the all research, was developed an IDS with ML. The results of the first model on a real infrastructure are presented in Fig. 11,12. At this research stage, the sixth version of the model has been developed and tested in SDN [20].

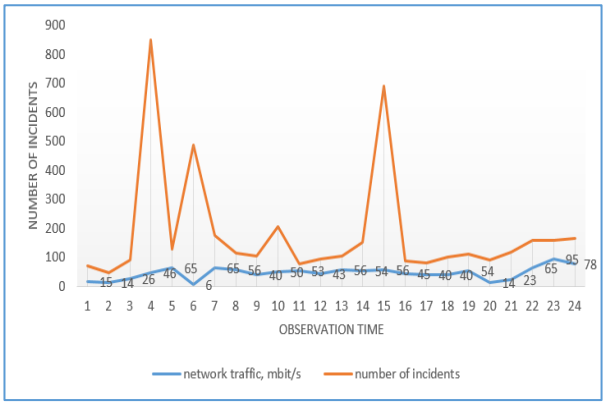

Fig. 11. Visualization of the work of the Snort IDS in a 24-hour period without a module with ML.

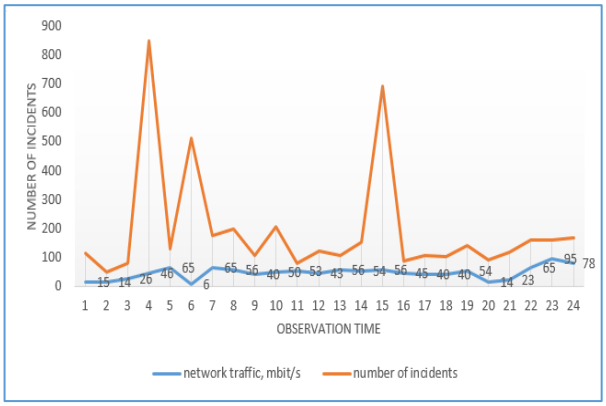

Fig. 12. Visualization of the work of the Snort IDS in a 24-hour period with a ML module.

Explanation of visualized results

The Fig. 5,6,7,8 present a visualization of the distribution of detected and classified malicious datasets embedded in network traffic at different iterations. The first and second iterations, the percentage of malware detection is about (7.6-8)%, the percentage of classification is less than 3%. The third iteration, the improvement in the solution of the detection problem is insignificant (7.9-8.02)%, but the solution of the classification problem becomes acceptable for practical use (14-16)%. An increase in the number of iterations on the same dataset leads to retraining of the RNN and an avalanche deterioration in the results of solving the problem of malware classification (Fig. 8). The most effective detection occurs at speeds up to 50-60 Mbps. The results of the work of the developed software integrated into the IDS Snort in various modes shows on Fig. 9,10. As can be seen from Fig. 9, 10, the use of a RNN at the level before the preprocessor increases the reliability of the data processed in the network IDS. An important factor when using a RNN before the preprocessor is the need for training datasets to differ not only quantitatively, but also variably.

Increase, in efficiency by (10-12)% managed to achieve only, the CTPH method.

8. Conclusion

The paper considers a software model for detecting malware using a RNN as part of the *Snort* version 2.9.18.1 IDS. A *pcap* network traffic file with embedded malware was used as a dataset. The training datasets for RNN are based on the source code of malware obtained from open sources. The *k nearest neighbors* method was used as a mathematical apparatus for solving the classification problem.

Based on the research, it can be concluded:

The use of the *k nearest neighbors* method at the *preprocessor level* is justified in the presence of a large and unique training dataset.

The use of augmentation for training a, RNN included in the IDS before the preprocessor is inappropriate, since solving the classification problem using the *k nearest neighbors* method requires a data set with unique data that differ from each other in many criteria, which is difficult to achieve using the augmentation method. The use of RNN as part of an IDS at the preprocessor level is justified in the presence of a large computing resource (a special role is played by the amount and type of RAM).

References

[1] G.Stoneburner, "*Underlying Technical Models for Information Technology Security*" , NIST Special Publication 800-33, 2001.

[2] R.Atefinia, M.Ahmadi, Performance Evaluation of Apache Spark Mlib Algorithms on an Untrusion Detection Dataset. [Online].Available:https://arxiv.org/abs/2212.05269

[3] M. Bachi, A. Harti, J. Fabini and T. Zseby, Walling up Backdoors in Intrusion Detection Systems. [Online].Available:https://arxiv.org/abs/1909.07866

[4] National standard of the Russian Federation, "Quality of official information", GOST R-51170-98, (2020)// 12, Moscow, Standardinform.

[5] B.E.Zolbayar et al, "Generating practical adversarial network traffic flows using NIDSGAN", [Online].Available:https://arxiv.org/abs/2203.06694
F. Zhong et al, "MalFox: Camouflaged adversarial malware example generation based on Conv-GAN againist black—box detectors", [Online].Available:https://arxiv.org/abs/2011.01509

[6] Dominik Kus et al, "A false sense of security? Revisting the state of machine learning-based industrial intrusion system", [Online].Available:https://arxiv.org/abs/2205.09199

[7] K.Jallad, M. Aljnidi and M.Desoki, «Big data analysis and distributed deep learning for next-generation intrusion detection system optimization», (2022)//[Online].Available: https://arxiv.org/abs/2209.13961

[8] A. Branitsky and I. Kotenko, «Analysis and classification of methods for detecting network attacks», Proceedings of SPIIRAS, (2016) // issue 45, pp. 207-244.

[9] Electronic resource dedicated to digital transformation technologies. [Online].Available:https://www.osp.ru/os/2020/03/13055601

[10] T. V. Jamgharyan and V.H.Ispiryan, "Network infrastructures assessment stability" *Proceedings of 13th International Conference on Computer Science and Information Technologies (CSIT)*, Yerevan, Armenia, pp. 199-203, 2021.

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

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

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

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

[15] G.Campos, A.Zimek, et al, «On the evaluation of unsupervised outlier detection: measures,datasets, and an empirical study». [Online].Available:https://link.springer.com/article/10.1007/s10618-015-0444-8

[16] Professional information and analytical resource dedicated to machine learning, pattern recognition and data mining. [Online].Available: http://www. machinelearning.ru

[17] T.Jamgharyan, “Research of obfuscated malware with a capsule neural network”, *Mathematical Problems of Computer Science*, vol. 58, 67–83, 2022.

[18] Website for identifying, defining and cataloging publicly disclosed cybersecurity vulnerabilities.
[Online].Available:https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-20685

[19] T.Jamgharyan, “Modernization of intrusion detection system via the generative model”, *«Haikakan Banak» («Armenian Army») Defense-Academic journal, National Defense Research University*, Ministry of Defense, Republic of Armenia, no. 2, pp.75-79, 2021.
[Online].Available:https://razmavaraget.files.wordpress.com/2022/01/hb2-final.pdf

Ներխուժումների հայտնաբերման համակարգի հավաստիության բարձրացման մոդելի հետազոտում

Թիմուր Վ. Ջամղարյան

Հայաստանի ազգային պոլիտեխնիկական համալսարան, Երևան, Հայաստան
e-mail: t.jamgharyan@yandex.ru

Ամփոփում

Հոդվածում ներկայացված են Snort 2.9.18.1 ներխուժումների հայտնաբերման համակարգի կազմում ռեկուրենտ նեյրոնային ցանցի կիրառման հետազոտության արդյունքները: Հետազոտությունն իրականացվել է athena, dyre, engrat, grum, mimikatz, surtr վնասաբեր ծրագրային ապահովման ելակետային կոդի հիման վրա *կառուցած* տվյալների հավաքածուներով: Շահագործվել է CVE-2022-20685 Snort ներխուժումների հայտնաբերման համակարգում խոցելիությունը: Մուտքային թրաֆիկի մշակումը իրականացվել է մինչ frag-3 և modbus պրեպրոցեսորները: Որպես մաթեմատիկական ապարատ օգտագործվել է k մոտակա հարևանների մեթոդը: Իրականացվել է ծրագրային ապահովման իրագործման մոդելավորում տարբեր կրկնություններում և արդյունքների արտացոլում: Հոդվածում չներառված հետազոտության արդյունքները հասանելի են https://github.com/T-JN կայքում:

Բանալի բառեր՝ մեքենայական ուսուցում, տվյալների հավաքածու, վնասաբեր ծրագրային ապահովում, k մոտակա հարևանների մեթոդը, ներխուժումների հայտնաբերման համակարգ, *CVE-2022-2068*:

Исследование модели повышения достоверности системы обнаружения вторжений

Тимур В. Джамгарян

Национальный политехнический университет Армении, Ереван, Армения
e-mail: t.jamgharyan@yandex.ru

Аннотация

В статье представлены результаты исследования применения рекуррентной нейронной сети для обнаружения вредоносного программного обеспечения в составе системы обнаружения вторжений *Snort*. Исследование проводилось на наборах данных сформированных на основе вредоносного программного обеспечения *athena, dyre, engrat, grum, mimikatz, surtr* с эксплуатацией в системе обнаружения вторжений *Snort* версии 2.9.18.1 уязвимости *CVE-2022-20685.* Обработка данных входного трафика осуществлялась *до препроцессоров frag-3* и *modbus*. В качестве математического аппарата использовался метод *k ближайших соседей*. Проведено моделирование работы программного обеспечения при разных итерациях и визуализация результатов. Результаты исследования не внесенные в статью представлены по адресу https://github.com/T-JN

Ключевые слова: машинное обучение, вредоносное ПО, метод ближайших соседей, система обнаружения вторжений, препроцессор, *CVE-2022-2068*.