УДК 004.056 https://doi.org/10.15407/pp2025.02.020

О.О. Летичевський, С.О. Євдокимов

ДВІЙНИКИ В СИСТЕМАХ ВИЯВЛЕННЯ ВТОРГНЕНЬ НА ОСНОВІ ГЛИБОКОГО НАВЧАННЯ

Дана робота спрямована на підвищення точності виявлення атак у програмних та апаратних системах шляхом використання цифрового двійника у формі алгебраїчної моделі в системах виявлення вторгнень (IDS), заснованих на нейронних мережах глибокого навчання (DNN). Цей підхід усуває недоліки навчання та недосконаліть набору даних, які призводять до численних помилкових спрацьовувань, невиявлених вторгнень та слабкої стійкості до атак супротивника. Пропонується архітектура IDS, яка поєднує нейронні мережі глибокого навчання з алгебраїчною моделлю на необхідному рівні абстракції. Ця композиція забезпечує високу точність виявлення та постійне самонавчання IDS на основі роботи моделі та збору даних, включаючи атаки нульового дня. Два приклади демонструють застосування цього підходу: виявлення атак у двійковому коді програмної системи та в програмованій інтегральній схемі.

Ключові слова: глибоке навчання, нейронна мережа, нейро-символьний підхід, цифрові двійники, алгебра поведінки

O.O. Letychevskyi, S.O. Yevdokymov

DIGITAL TWINS IN INTRUSION DETECTION SYSTEMS BASED ON DEEP LEARNING

This work aims to improve the accuracy of attack detection in software and hardware systems by utilizing a digital twin in the form of an algebraic model within intrusion detection systems (IDSs) based on deep learning neural networks (DNNs). This approach addresses the shortcomings of training and dataset imperfections that lead to numerous false positives, undetected intrusions, and weak resistance against adversarial attacks. We propose an IDS architecture that combines deep learning neural networks with an algebraic model at the required level of abstraction. This composition provides high detection accuracy and enables continuous self-learning of the IDS based on model operation and data acquisition, including zero-day attacks. Two examples demonstrate the application of this approach: detecting attacks in the binary code of a software system and in a programmable integrated circuit.

Keywords: deep learning, neural network, neurosymbolic approach, digital twins, behavior algebra

Introduction

The modern intrusion detection systems (IDSs), which are based on deep learning neural networks (DNNs), have benefits and shortcomings. The detection of attacks in real time is carried out by analyzing incoming traffic, from which it is possible to extract some traffic, statistical and time data.

The indisputable advantage of DNNs is that they can work in real time, quickly detecting intrusions. In general, IDSs can be divided into two functional groups [1].

The first variety refers to classifiers trained on data labeled with specific types of attacks or multiclassifications. Everything that does not fall under this dataset is considered benign traffic.

The second group is trained on examples of normal system operation, and any deviation is considered an intrusion. These systems are called anomaly-based systems.

Because datasets are not able to cover the full space of possible data, cases of false positives can arise. Moreover, a hacker can evade classification by manipulating sensitive features fed to the DNN input.

Modern intrusion detection systems work for both software and hardware systems, including those based on programmable

integrated circuits, namely field programmable gate arrays (FPGAs) that are very popular in the Internet of Things environment.

In software systems, IDSs are implemented in the form of firewalls that contain a neural component and block malicious traffic. Often, systems are created based on FPGA or application-specific integrated circuit (ASIC), which directly contain neural networks [2].

The internet protocols TCP/IP and UDP, from which the traffic is considered, are mostly encrypted data, so only available data from traffic packets and statistical and time data are generally used. It reduces the accuracy of classification as well.

There are a certain number of methods that attempt to eliminate such disadvantages, particularly the use of sandbox technology. However, running the application in an isolated environment significantly slows down traffic and cannot be used effectively in real time.

The authors have developed technology in which there is a digital twin [3] of the system to be protected. This notion is known as an established concept in various subject domains to study and analyze the original system. The digital twin is presented in the form of an algebraic model created at the appropriate level of abstraction for effective intrusion detection. We consider the architecture of the system, where both types of DNNs are involved, which interact with the digital twin.

The system also uses the augmentation of the dataset using the generation of new features via a digital twin. For this purpose, a limited initial run is performed in the digital twin environment to obtain the relevant data.

1. General scheme of the approach

The main idea behind the IDS approach proposed by the authors is to combine an algebraic method, which provides accuracy in detection, with a neural network of deep learning. This approach is considered neurosymbolic. This approach is used in practice and has already produced serious results that have significantly increased the accuracy of the DNN [4].

To implement the symbolic or algebraic component, a digital twin of the system is created in terms of algebraic specifications, and this twin is stored with the system and interacts with the DNN. One of the functions of the digital twin is to create constraints for confirming or refuting the classification result of the neural component.

The neural component extracts the relevant features from the traffic and provides classification. It interacts with a digital twin and retrains according to this interaction.

The neural component is a composite of two DNNs, one of which is trained for binary classification—that is, determines normal operation and intrusion. The other works in parallel and classifies the type of attack.

The front-end component perceives the traffic from which it extracts the relevant features and, when interacting with the digital twin, receives additional features from the model. These data are also used to retrain the neural components.

Constraint-matching component computes the truth value of the constraints according to the traffic data and then makes a final verdict on the presence of an attack taking into account the DNN classification.

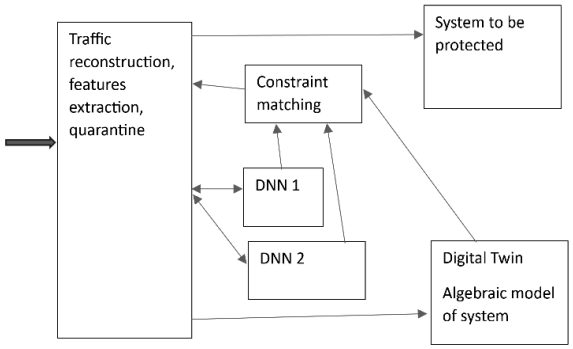

Fig. 1. General scheme of the approach

The algorithm for such a configuration is as follows:

1. A portion of the traffic is received, which extracts the relevant features of the

front-end component. Traffic is blocked in quarantine.

2. The portion of traffic is classified by both DNNs.

3. If both DNN components do not detect malicious traffic, then the traffic data are passed.

4. If at least one of the components detects an intrusion, the necessary features are submitted to the matching component with constraints data base:

a. If the matching with the constraints reveals an attack, then the intrusion is confirmed. Moreover, if one of the DNNs does not detect an attack, the received data are used for retraining.

b. If the matching with the constraints does not reveal an attack detected by the DNN with binary classification, then such an attack is considered unknown, and the traffic data are modeled on the algebraic model (digital twin). If the data lead to a violation of security properties, then the corresponding constraints are added to the constraint matching component, and both DNNs are retrained on the new case. All known security properties—such as unauthorized access, writing to forbidden memory, overflow, and others—are determined in advance and formalized as algebraic specifications.

2. The digital twin of the system as an algebraic model

The algebraic twin of the system is its model created via algebraic specifications. The basis of specifications is the algebra of behaviors [5], which is defined by operations and predicates on a set of actions and behaviors of different agents and environments. In turn, agents and environments are formal entities defined by a set of typified attributes. As such, we consider a system that is represented by its binary code, that is, a set of machine instructions for software systems or FPGA boot code for hardware.

The algebraic model is obtained automatically via appropriate translation.

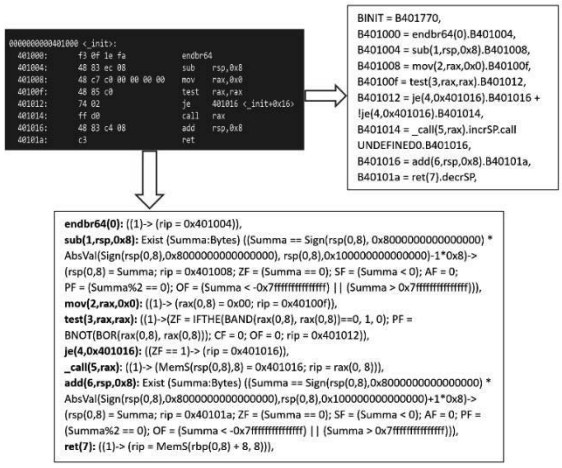

Fig. 2. Example of translation of binary code into algebraic specifications in the form of behavioral equations

Algebraic specifications are represented by the algebra of behaviors, which defines operations on the actions and behaviors of interacting agents. The operation "." is a prefixing operation that separates action from behavior. The "+" operation defines a nondeterministic choice of behaviors. Algebra is extended by operations of parallel and sequential compositions of behaviors. The behavior of the system is a set of equations in the right part of which there is an expression in terms of the algebra of behaviors. Action is a pair: a precondition that defines the state of the environment in which the action is possible and a postcondition that defines a change in the environment if the action is performed.

Thus, Figure 2 shows a set of machine instructions, which are translated into the equation of the algebra of behaviors and into the set of corresponding actions. In fact, the equations of the algebra of behaviors represent the control flow of the program, and the actions contain the semantics of the corresponding machine instruction.

For hardware, translation of the executable FPGA code is used, which is obtained after compiling the code from the hardware design language. In particular, this code is contained in the programming object file (pof).

The corresponding semantics of the commands of this file can be presented in the form of equations of the algebra of behaviors.

With an algebraic model, it is possible to apply formal methods, particularly algebraic modeling, to find vulnerabilities or possible attack scenarios.

Algebraic modeling is used to determine the reachability of some property. If the initial state of system S_0 is determined as a set of constraints F or specific values of system attributes $x_1, x_2, \ldots$

$$S_0 = F(x_1, x_2, \ldots) \quad (1)$$

Then, when choosing a possible action from the equations of the algebra of behaviors, we check the satisfiability of its precondition and environment formula, and if there are such values of attributes for which the formula is true, then we change the state according to the postcondition. Thus, during algebraic modeling, we obtain a set of symbolic states S_0, S_1, ..., namely set of formulas where the values of the attributes are not concrete; that is, each symbolic state covers a set of concrete states.

If the sought-after property $R(x_1, x_2, \ldots)$ is achievable, then there is a sequence of symbolic states that leads to the S_R state that the conjunction of the state and property is a satisfiable formula; that is, there are values $x_1, x_2, \ldots$ such that $S_R \wedge \mathrm{R}(x1,x2,\ldots)$ is true.

In addition to algebraic modeling, other formal methods can be applied, such as the search for state invariants or proving the completeness of the system (absence of deadlocks) and other useful procedures.

In the context of cybersecurity, we consider the conditions for the possibility of an attack detection or the search for vulnerabilities, which are specified by formulas over the attributes of the system, as the sought-after properties.

The algebraic twin of the system works in parallel with the neural component and performs the following functions:

1. Generates a set of constraints for classifying attacks. The generation method is described in the next section.

2. An augmented training dataset is generated based on a digital twin initial state to obtain the additional features.

3. A set of cases is generated for training a binary neural network. Generation is performed via algebraic modeling, here with the help of which possible scenarios of benign behavior of the system are built and from which appropriate features for training are extracted.

4. System behavior modeling is performed to identify an unknown attack.

5. The possible use of a digital twin is the implementation of a deceptive strategy. If the intrusion is for the purpose of reconnaissance or data theft, the digital twin can generate responses with fake data to give to the intruder. This property is not considered in the present work but is the subject of future research.

3. Attack patterns and generation of constraints

Using the algebra of behaviors, it is possible to describe the semantics of the attack. We call this pattern an algebraic attack signature.

This signature is determined by the equations of the algebra of behaviors, in which unknown or arbitrary behavior is involved. In the actions, additional constraints for triggering the attack may be present in the precondition.

The formal definition of an algebraic signature is as follows.

Algebraic signature B_i $(x_1, x_2 \ldots, Y_1, Y_2, \ldots)$ is a behavioral equation over the attributes of the system $x_1, x_2, \ldots$ and arbitrary behaviors $Y_1, Y_2 \ldots$, which determines the behavior of the intruder in the environment of the model, which is specified by the system of behavioral equations.

An attack is achievable in the model environment if there is a sequence of states that corresponds to a given algebraic signature of the attack. This correspondence is verified via algebraic modeling, particularly its variant, that is, the algebraic matching algorithm [6].

This sequence of states is used to generate constraints that are used in the intrusion detection system. If there is a sequence of states that leads to the triggering of a vulnerability or an intrusion $S_0, S_1, S_2, \ldots$, then we can to specify a constraint from the formula S_0that corresponds to a given attack.

In this case, it is necessary to strengthen the formula S_0 such that any set of attributes corresponding to this formula triggers the attack.

This is accomplished via backward algebraic modeling, where the initial state is defined as the state at which the attack is triggered and through the actions are performed for the given sequence of states leading to the initial state. The obtained formula of the initial state at which the attack is triggered is used as a constraint of the intrusion detection system corresponding to this attack.

When an attack is detected, it checks whether the constraint is true on a set of features that have been extracted from the traffic.

Note that this technology is possible when features that correspond to system attributes are used. We use a digital twin to identify such features and augment the training dataset.

Below, we consider specific examples of constraint generation and dataset augmentation.

4. Case Study

4.1 Stack Corruption Attack in a Software System

Consider the algebraic signature of the "stack corruption" vulnerability. This is a simple example of a vulnerability that can be exploited by an intruder by inputting specific data. The signature is considered as reduced only to illustrate the method.

We can describe this vulnerability via a behavioral equation:

StackDamage = syscall(0). Z; mov(ss, y, z, regn)

This *StackDamage* equation represents behavior that begins with the *syscall* action, continues with the arbitrary *Z* behavior, and ends with the *mov* action. Each action corresponds to an Intel x86 machine instruction.

The semantics of the *system* action with the specified parameters indicate an interruption when it comes to inputting information from a file. This defines the processor register *rax*, and the memory address to which the information will be written is in the *rsi* register. This memory is additionally labeled in the postcondition as *Dirty* or possibly harmful information. This will be taken into account during algebraic modeling.

syscall(0) : (rax(0,8) == 0)-> (Dirty(rsi(0,8)) = 1)

The final *mov* action contains the vulnerability itself, that is, a write outside the stack boundary in the stack memory. Additional action semantics are defined as follows:

mov(x, y, z, r) : (y+z >= top(0,8)) -> 1

The action parameters recognize x – memory type, in this case stack memory, y – stack address, z – memory size, and r – general purpose register from which the memory is written. The notation *top (0,8)* indicates the current top of the stack or the address of the stack pointer.

This signature is used in the algebraic matching algorithm, where the system model and this signature are input. Using the method of algebraic modeling, it is proven that this behavior is reachable. As a result, we obtain the final state in which the vulnerability is triggered.

This state contains constraints on system attributes such as register values and memory. Registers can be both stack and general purpose.

$$Mem(x) >= 0xffffffff \wedge Reg1 > 0xf001 \wedge SP == MEM\_LIMIT - 0x00ff \wedge \ldots$$

Next, it is necessary to determine the initial values of system attributes in the form of constraints under which the vulnerability is triggered.

We use backward modeling from the final state to the point where the input occurs via a syscall instruction.

As a result of backward modeling, we obtain a state that determines the constraints on the attributes under which the vulnerability can be triggered. For example, with the selected modeling, we determine the constraint for the initial memory, which is located at the address recorded in the rsi register. For example:

$$Memory(rsi) > 0xffffffffffff0011$$

Thus, when such data are entered from a file, a vulnerability is triggered, and if such data are present in the traffic, the program stack will be damaged and the system will stop working. Thus, we can determine the exploit of this vulnerability by defining one of the memory values that will be specific, for example, *0xffffffffffff00ff*.

The memory and register values may be given as the features that can be obtained from a partial initial run of the digital twin. The constraints related to such features are subsequently entered into a special database and matched with the appropriate classification of the neural component.

4.2 Violation of the Security Properties of Hardware

For hardware, the architecture that will make the digital twin effective has not yet been finalized. The model that exists in computer resources with access to traffic data in hardware is considered.

When hardware systems are designed, the design languages use local constraints or assertions, the violation of which is considered a security violation. Proving the impossibility of violating such a constraint, that is, invariant, is quite difficult because parts of the hardware design work mainly in parallel; therefore, the search can be endless or take considerable time.

Let us have a security property $R(x_1, x_2, \dots)$ at a certain location of the hardware system, which can be obtained already in the code that is loaded. It can be either password cracking or data overflow. We have a model of the system in terms of the algebra of behaviors. It is a translated from set of commands contained in .pof files for loading onto the FPGA.

We believe that we have a DNN that is trained to classify violations of security properties in the hardware system.

In this case, the generation of constraints can occur not according to the algebraic signature but as a result of modeling from the point of security violation $\neg R(x_1, x_2, \dots)$ before the start of receiving FPGA data from the external environment.

We obtain a set of scenarios from the modelling for values of the input signals. Also we obtain a set of constraints according to the behavior scenarios. The resulting constraint that is used will be the union of all the constraints at the data entry point.

In general, detecting attacks in hardware will be performed as in a software system. In addition to specific attacks, constraints for well-known attacks, such as side channel attacks, can be added.

Discussion and conclusions

We used a neurosymbolic approach in intrusion detection systems. In accordance with this architecture, several experiments were conducted using two DNNs and an algebraic component. DNNs were trained on available datasets [7] and augmented with features extracted from the limited initial launch of the digital twin. Data from registers and memory, which receive data from traffic, were used. For this purpose, an experimental prototype was developed, which uses standard traffic reconstruction frameworks and provides a limited initial launch of a digital twin that works with the DNN composition.

To evaluate the accuracy, a test dataset labeled with different types of attacks was selected. The first classification results were obtained in which there were no false positive detections, which confirmed the proof of concept.

The experiment was conducted on the separate use of components, which are planned to be integrated in the future into a single platform to create a software product.

In this architecture, the digital twin performs the function of generating constraints, which are used to confirm the classification of the DNN. Here, constraints are used as possible exploits in software or hardware systems to exploit a particular vulnerability. It can be assumed that the programmer wrote a program without vulnerabilities or used tools to verify vulnerabilities. On the other hand, compilers and operating systems have protection against classic vulnerabilities, such as writing to unauthorized memory. The program author cannot be certain that the compiled code that uses the interaction of different libraries does not contain vulnerabilities because vulnerabilities can be caused by the incorrect use of libraries, incompatibility of resources, and unknown paths that lead to violations. This especially applies to hardware, where there is a high degree of parallelism of various system components, which makes exhaustive verification impossible. When verifying the vulnerabilities of large systems, even at the level of binary code, different sequences of actions encounter a combinatorial explosion in the search space. Therefore, even before launching the system into permanent operation, it is impossible to exhaustively verify everything, and one of the functions of a digital twin is the permanent search for

vulnerabilities in the system and generation of appropriate constraints.

Although this architecture has a serious advantage in accuracy, several shortcomings must be considered.

We assume that a binary neural component that determines deviations from the normal operation, here trained on a certain amount of benign data, will always correctly determine the normal operation, but because it is impossible to cover the entire space of benign scenarios, false positive detections are possible. This is not an indisputable fact because a hacker can adapt to a normal scenario, even though this is quite a difficult task and requires studying the work of the classifier system that the hacker is likely to use and may be a case of not detecting the intrusion.

In this configuration, the result of detecting an attack by a binary neural component and not confirmed in the database of constraints must be verified by modeling, which determines the possibility of the reachability of the security property violation. This can cause traffic delays and system disruptions.

Thus, the presented architecture is an indisputable step in increasing the accuracy of intrusion detection systems based on the neurosymbolic approach, where the synergy of the speed of classification by the neural component and the accuracy of the algebraic approach is used. A certain number of vulnerabilities and attacks have been investigated, but a longer formalization and extension of the database of constraints and training of the system is appropriate for large critical systems that need to be protected. The optimal architecture of a digital twin for hardware has not yet been determined, so further experiments and more experience in finding vulnerabilities in hardware are required.

Acknowledgment

This work was carried out with the support of the project “Application of Algebraic Approach and Enhanced Artificial Intelligence Methods in Cybersecurity Tasks” within the NATO program “Science for Peace and Security Program.”

References

1 A. Pinto, L.-C. Herrera, Y. Donoso, and J. A. Gutiérrez, “Survey on intrusion detection systems based on machine learning techniques for the protection of critical infrastructure,” *Sensors*, vol. 23, no. 5, p. 2415, Mar. 2023, doi: 10.3390/s23052415.

2 D.-M. Ngo, A. Temko, C. C. Murphy, and E. Popovici, “FPGA hardware acceleration framework for anomaly-based intrusion detection system in IoT,” in *Proc. 31st Int. Conf. Field-Programmable Logic Appl. (FPL)*, Dresden, Germany, 2021, pp. 69–75, doi: 10.1109/FPL53798.2021.00020.

3 S. Haag and R. Anderl, “Digital twin—proof of concept,” *Manufacturing Letters*, vol. 15, pp. 64–66, Jan. 2018.

4 P. Hitzler, A. Eberhart, M. Ebrahimi, M. K. Sarker, and L. Zhou, “Neuro-symbolic approaches in artificial intelligence,” *National Science Review*, vol. 9, no. 6, Jun. 2022, doi: 10.1093/nsr/nwac035.

5 A. Letichevsky, “Algebra of behavior transformations and its applications,” in *Structural Theory of Automata, Semigroups, and Universal Algebra*, V. B. Kudryavtsev and I. G. Rosenberg, Eds. Dordrecht, The Netherlands: Springer, 2005, pp. 241–272.

6 O. Letychevskyi and V. Peschanenko, “Applying algebraic virtual machine to cybersecurity tasks,” in *Proc. 2022 IEEE 9th Int. Conf. Sci. Electron., Technol. Inf. Telecommun. (SETIT)*, Hammamet, Tunisia, 2022, pp. 161–169, doi: 10.1109/SETIT54465.2022.9875895.

7 Y. Mirsky, “Kitsune network attack dataset,” Kaggle. [Online]. Available: https://www.kaggle.com/datasets/ymirsky/network-attack-dataset-kitsune. [Accessed: 14.09.2024].

Одержано: 18.03.2025
Внутрішня рецензія отримана: 27.03.2025
Зовнішня рецензія отримана: 28.03.2025

Про авторів:

[1]*Летичевський Олександр Олександрович,*
доктор фізико-математичних наук,
завідувач відділу
https://orcid.org/0000-0003-0856-9771

[2]*Євдокимов Сергій Олександрович,*
аспірант кафедри комп'ютерних наук та
програмної інженерії
https://orcid.org/0000-0001-7213-0259

Місце роботи авторів:

[1]Інститут кібернетики імені В. М.
Глушкова НАН України,
03187, м. Київ,
просп. Академіка Глушкова, 40.
Тел. (+38) (044) 526-20-08
Email: office@icyb.kiev.ua,
www.incyb.kiev.ua

[2]Херсонський державний університет
73003, м. Херсон (Юридична адреса),
вул. Університетська, 27.
76018, м. Івано-Франківськ (Фактична
адреса), вул. Шевченка, 14.
Тел. +38 (0552) 226263
Email: office@ksu.ks.ua