DOI 10.20535/2411-1031.2020.8.1.218004
UDC 004[942::413.4]

MISNIK OLEKSII

APPLICATIONS CONTAINERS SECURITY MODEL

It has been established the purpose of container environments for the development, delivery and operation of various types of the software applications. The web and mobile applications have the most widespread use. This is due to the container media's emphasis on quick loading and installation. Using this method, you can think of the infrastructure as a code and get the benefits associated with it. First of foremost, accelerate the development of software applications, particularly reducing the time between their conception and launch. This is facilitated by the use of download utilities, the deployment of container environments on container virtualization platforms, and the management of software applications. Despite this, the necessity to secure the security of software programs limits the adoption of container systems in practice. This is primarily due to the use of standard approaches based on intrusion detection systems. Features of container environments in relation to real settings were overlooked when they were first introduced. Taking into account the vulnerabilities and dangers of container virtualization platforms, as well as monitoring the processes of container environments given the unique architecture and input load flow, it is important to keep in mind that there are only a few of them. A model for assuring the security of container environments of software programs is proposed to overcome the difficulties of employing intrusion detection systems. It is based on the idea of using system calls of the host system on the example of the Linux operating system. This is because they allow the software applications to interact with the kernel. As a result, users have been identified as the sources of probable intrusions into container environments. Additionally, there are examples of atypical commands for analysis during the execution of system calls. Based on the obtained results, it has been distinguished the stages of intrusion detection and transitions between them. As a result, the Petri net is used to formalize this process. During the intrusion detection, it has been defined by the numerous sets of stages, transitions between stages, relations between stages, and transitions.As a result of the suggested approach, the security aspects of container environments for software applications are possible to be established.

Keywords: application, container, security, intrusion detection, intrusion detection system, system call.

Problem statement. Container environments are intended for the creation, delivery, and operation of numerous types of software programs [1]. The most widespread use of both online and mobile applications is among these types. This is because the development of container environments has focused on making them easier to load and install [2], [3]. It is feasible to separate the infrastructure and, as a result, manage its computer resources by analogy with the program using this technique. Because of this possibility, infrastructure as code (Infrastructure as Code, IaC) [4] has been considered. As a result of container environments, the development of software applications, notably writing and testing, is also expedited. Therefore, the period between writing and launching them is shorter. These advantages pertain to the deployment of container environments on container virtualization platforms, as well as the use of utilities to download and manage software programs, and, as a consequence, the optimization of server computing resources [3], [5].

However, despite these benefits, the practical implementation of container environments for software applications is constrained by the necessity to assure their security. This is primarily due to the usage of standard approaches based on intrusion detection and prevention systems. Features of container environments' application in comparison to physical settings were overlooked when they

were first introduced.Among them are taken into account the vulnerabilities and dangers of container virtualization platforms, as well as monitoring the processes of the container environment given its unique design and input load flow. Increase the calculated capabilities of server equipment as necessary as the load on it grows [6].

Therefore, to overcome the challenges of employing systems to detect and prevent intrusions while also taking into account the unique characteristics of container environments, it is proposed to ensure their security based on system calls to the operating system. The primary idea behind this hypothesis is to keep the functioning container environment visible without modifying its architecture (for example, the image of the container, preload libraries, program code). System calls will be logged at the kernel level of the operating system [7]. Therefore, ensuring the security of container environments of software applications is important.

Analysis of recent research and publications. The analysis of security challenges utilizing intrusion detection systems is discussed in [8] - [16]. [8] - [9] provide a generalized description of this process. [10] reveals the necessity of fulfilling the need to identify breaches, particularly in the financial sector. While [11] - [15] focus on the solution of security issues in container settings for software applications. These sources, in particular, support the use of intrusion detection systems. The examination of available sets of rules for detecting possible network assaults [11] is one of the difficulties that has been solved. The use of intrusion detection systems is reduced to notification of atypical activity in this instance. This is accompanied by issues of incompatibility of the established rules. Furthermore, it is possible to add here the usage of the signature method, which necessitates the creation of distinct rules to identify new invasions. In [12] features of formalization of the intrusion detection method are discussed. In particular, attention is paid to the collection and analysis of information at the application level from specific software applications, such as databases, web servers. Intrusions at other levels, such as channel, network, and transport, may not be considered as a result. [13], [14] describe real-time processing of massive data streams and intrusion detection. In particular, there is a demand on the server equipment's CPUs, especially at high loads. This limits the applicability of this method to container media safety. [15], [16] describe architectural aspects of the use of intrusion detection systems, including the requirement for and drawbacks of decoding traffic for analysis. Among the disadvantages are, for example [16], where high expenses while decrypting SSL, and sensitivity to packet loss.

Therefore, employing intrusion detection systems to ensure the security of container environments for software applications is complex due to the enormous number of factors and actual properties of these environments. For starters, this is owing to the ability to use a lot of operating systems at the same time. As a result, it is linked to a number of loading options, distinguishing access to container environments. Furthermore, a wide range of software should be included among the features. Since these characteristics influence the feasibility and, in particular, the detection of breaches, they are established by constructing a container security model based on system calls.

The aim of this paper is to determine the security aspects of container environments for software programs that rely on operating system calls.

The main material research. A container in a network protocol model is an abstraction at the application level. The packaging of code and related dependencies for downloading software applications characterizes it. This is accomplished by employing a separate operating system instance. A container is a series of containers that forms an isolated environment containing software programs. Fig. 1 [17] shows an example of a representation of its architecture, namely:

1. Physical equipment consists of the server hardware that is used to install and run the operating system.
2. Host system is an operating system (for example, Linux) that distinguishes between containers the technical features (for example, RAM) of server equipment on a logical level.
3. Virtualization software is a client-server application that loads the container image and constructs the container's environment.
4. Container is a secure environment in which applications run on their own Linux OS.

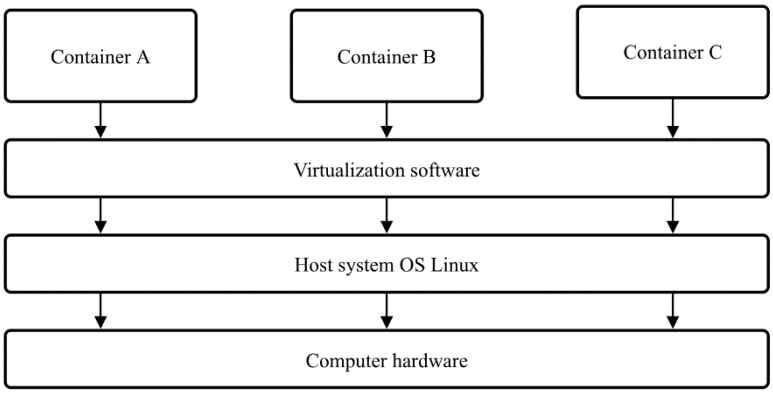

Figure 1 – Representation of the architecture of the container environment of software applications

We outline the functions of chosen components of the container environment of software programs, which are challenging to implement using virtualization approaches, particularly [18]:

1. Containers share the primary host system's physical resources, with a logical allocation between them (for example, RAM,the hard disk memory).
2. The unification of container creation and start eliminates software faults [19] caused by changes in container environment setup for normal application deployment.
3. You can emulate the operation of distributed computer systems by loading multiple containers at the same time.
4. Containers are designed to suit the needs of end users and developers without the need for cloud-based software deployment. This eliminates the issues that come with different user environment setups.

From a security standpoint, we will have a look at the first purpose of container environments on the list, which is the utilization of physical resources on the main host system. Selecting this option allows you to choose at what point in the interaction between the container and the host system system calls can be tracked and information about probable intrusions can be obtained. This is accomplished by making the kernel of the operating system's fundamental component [20]. This ensures that software applications have coordinated access to the host system's resources (for example: processor time, memory and hardware). The kernel also provides file system and network protocol support. It is the lowest level of abstraction of access to physical resources and is the main component of the operating system. At the same time, the kernel gives access to the processes of the corresponding software programs through a method of process interaction and access to the operating system's system calls. This means that any activity in the container is at first represented by a system call and delivered to the host system's kernel, and then it launches a process based on the host system's kernel policies.

Since the host system's core interacts directly with the hardware, software applications are isolated from the architecture's features. In addition, the kernel performs input/output tasks (opening, reading, writing, and managing files), as well as the creation and management of processes, and their synchronization, and interaction. These services are available to the software applications via system calls (Fig. 1). This explains why system call containers are used to communicate with the host system. As a result, we can predict that the fewer system calls are available, the lower are intrusions into container environments [21]. The independence from a range of software applications is a distinguishing aspect of such connection with the host system's core. The reason for this is that system calls define the format of the kernel service requests. The process initiates a request for a specific kernel procedure that is similar to a typical library function call. The kernel executes the request on the process's behalf and returns the required data to it.

In container settings, all software programs will be needed to use system calls (see Fig. 2). They are made to interact directly with the kernel of the host system, which operates in a privileged mode and has access to the system tables, external device ports, and the memory manager (Fig. 3).

User space commands are executed in the kernel of the host system (for example, Linux) through system calls [22]. So let's separate the information about system calls, which can be intercepted due to the standard functionality of the Linux operating system, and try to understand the actions of software applications in case of intrusions. To do this, let's select users, each of which can be considered as a source of probable intrusion and, as a consequence, try to consider their characteristics:

– a system administrator with remote access. In this case, connecting the system administrator to the container environment and performing actions in it is a special case. This is due to the fact that all new installations are due to automated deployment systems. A characteristic feature of such access is authorization and formalization over time. Therefore, when using system calls with unusual metadata that will look like malicious data, the administrator has the opportunity to mark them as a known false positive;

– running of a software application. In this case, it does not need to use system calls with unusual metadata. However, even if someone uses this action, it is possible to add them to the whitelist with the mandatory addition of metadata. They will determine that the challenge is not an invasion;

– the actions that should take place if the host system vulnerability is exploited. In this case, access is not authorized and the execution of system calls with unusual metadata is considered an intrusion. This means that by checking the kernel of the host system for the execution of atypical commands or system calls with atypical metadata, we can make assumptions about the beginning of the invasion.

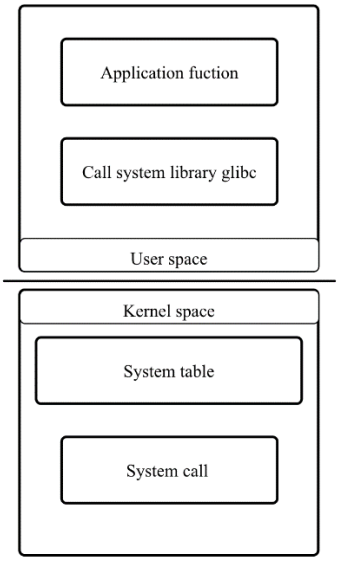

Figure 2 – The software application's interaction with the kernel of the host system

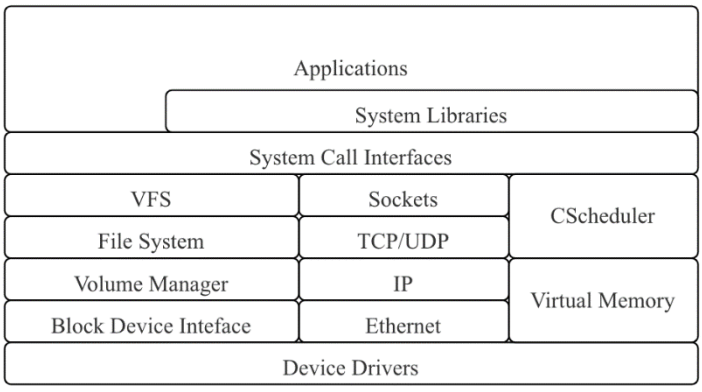

Figure 3 – A list of system calls that can be used to access application functions

Here are some examples of atypical command execution that should be examined during the syste call execution on the host system (on the example of the Linux OS) [23]:

- increased privileges through the usage of privileged containers;
- reading or writing of the service directories, such as / etc, / usr / bin, / usr / sbin;
- the symbolic links;
- the changes of rights of file access;
- an unexpected network connections;
- the process of creation via execve execution of binary file shells, such as sh, bash, csh, zsh;
- an execution of SSH binary files (such as ssh, scp, sftp) during non-working hours;
- the minimal shadow-utils or password modification.

As a result, when software programs interface with the host system's core via system calls, you can influence a wide range of parameters. It is possible to detect indicators of invasion in its early stages behind them. As a result, when intrusions are identified, the use of system calls is accompanied by certain states and scenarios, particularly transitions from one condition to another. System calls, regardless of software applications, follow the same algorithm, implying that they complete the same steps in both ordinary and abnormal (for example, intrusion-related) operations. In the latter case, when vulnerabilities are exploited, they result in invasions and, as a result, access to the host system's resources. When the characteristics of system calls are examined in depth, it is possible to characterize the stages and dynamics of software application invasion and detection in the container environment (Table 1).

Table 1 – Intrusion detection stages based on system calls

Stages of intrusion detection		Transitions between levels of intrusion detection	
p_0	Call of the applications to the kernel	t_0	Reading of the event
p_1	The kernel reads the event	t_1	The event execution
p_2	The event is detected	t_2	Recognizing of the kernel-level event
p_3	The event message is sent	t_3	Sending the event for its validation
p_4	The event has been received for its further validation	t_4	Creating the event messages
p_5	An event check request has been made	t_5	The event type recognition
p_6	Checking of the event arguments	t_6	Performing of the event analysis
p_7	Probable intrusion	t_7	The report generation
p_8	The probable intrusion analysis	t_8	The report submission
p_9	Reporting of the analysis results	t_9	The report formation
p_{10}	The report transmission	t_{10}	The report formation
p_{11}	The message is generated	t_{11}	Sending a message
p_{12}	The message is sent	t_{12}	The message processing
p_{13}	The message is received	t_{13}	The conclusion formation
p_{14}	The message is recorded	t_{14}	The data logging

It is feasible to formalize the process of detection of intrusions into the container environments of software programs based on the stages and dynamics of the invasions. We employ the Petri nets [24] as a mathematical tool to accomplish this. This is owing to the fact that they may be used to describe dynamic discrete systems that are made up of a series of parallel interacting processes. After that, let's consider the table. The following is the definition of the Petri net:

$$G = \{P, T, F\},$$

where P – is the set of intrusion detection stages, $P = \{p_i\}, i = \overline{1,14}$;

T – is the set of transitions between intrusion detection stages, $T = \{t_j\}, j = \overline{1,14}$;

F – is the incidence function, which shows the link between the elements P and T, for example, Fig. 4.

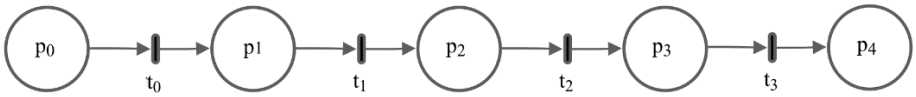

Figure 4 – A portion of a picture depicting the Petri net's detection of incursions into the container environment of software applications

Conclusions. As a result, the idiosyncrasies of their utilization play a major role in maintaining the security of container environments for software applications. For example, the shared use of the host system's physical resources, the centralized construction and launch of software applications in container environments, and the simultaneous operation of multiple containers on a single host system are all examples. They make it more difficult to employ typical methodologies based on intrusion detection systems to ensure the security of container environments for software applications. Taking into consideration system calls overcomes this issue. This helps you to detect invasions while they are still in the planning stages.In the spotlight of this, it has been provided a methodology for assuring the security of software application container environments. It is based on the Petri nets mathematical equipment. This is owing to the fact that they may be used to model dynamic discrete systems. As a result of the suggested model, it will be feasible to determine the security aspects of software application container environments.

REFERENCE

[1] Best Practices for Running Containers and Kubernetes in Production. [Online]. Available: https://www.gartner.com/en/documents/3902966/best-practices-for-running-containers-and-kubernetes- in-. Accessed on: Dec. 14, 2019.

[2] 2019 Container Adoption Survey. [Online]. Available: https://portworx.com/wp-content/uploads/2019/05/2019-container-adoption-survey.pdf. Accessed on: Dec. 14, 2019.

[3] D. N. Tyazhelnikov, P. A. Tokarev, and I. D. Petrov, "Virtualization of the workspace with the acceleration of 3D applications on the server side using Docker", *Problems of Modern Science and Education*, no. 14, pp. 21-23, 2017.

[4] Infrastructure as Code. [Online]. Available: https://infrastructure-as-code.com/. Accessed on: Dec. 14, 2019.

[5] A. R. Sampaio, J. Rubin, Beschastnikh, N. S. Roca, "Improving microservice-based applications with runtime placement adaptation", *The Journal of Supercomputing*, vol. 10, no. 4, pp. 1-30, 2019, doi: 10.1186/s13174-019-0104-0.

[6] A. Milenkoski, K. R. Jayaram, and S. Kounev, "Benchmarking Intrusion Detection Systems with Adaptive Provisioning of Virtualized Resources", in *Self-Aware Computing Systems*, pp. 633-657, 2017, doi: 10.1007/978-3-319-47474-8_22.

[7] I. Rosenberg, and E. Gudes, "Evading System-Calls Based Intrusion Detection Systems. Network and System Security", in *Proc. International Conference on Network and System Security*, Taipei, Taiwan, 2016, pp. 200-216, doi: 10.1007/978-3-319-46298-1_14.

[8] National Institute of Standards and Technology. (2007, Febr. 20). *NIST SP 800-94, Guide to Intrusion Detection and Prevention Systems*. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-94/final. Accessed on: Dec 10, 2019.

[9] International Organization for Standardization. (2015, Febr. 11). *ISO/IEC 27039, Information technology. Security techniques. Selection, deployment and operation of intrusion detection and prevention systems*. [Online]. Available: https://www.iso.org/standard/56889.html. Accessed on: Dec 10, 2019.

[10] PCI Security Standards Council. (2018, May 01). *Payment Card Industry Data Security Standard.* [Online]. Available: https://ru.pcisecuritystandards.org/_onelink_/pcisecurity/ en2ru/ minisite/en/docs/PCI_DSS_v3_2_RU-RU_Final.pdf. Accessed on: Dec 10, 2019.
[11] M. Aldwairi, A. M. Abu-Dalo, and M. Jarrah, "Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework", *EURASIP Journal on Information Security*, 2017:9, 2017, doi: 10.1186/s13635-017-0062-7.
[12] V. Mishra, V. K. Vijay, and S. Tazi, "Intrusion Detection System with Snort in Cloud Computing: Advanced IDS", in *Proc. of International Conference on ICT for Sustainable Development*, Washington, USA, 2016, pp.457-465.
[13] A. Belova, and D. Borodavkin, "Comparative analysis of intrusion detection systems", Actual problems of aviation and astronautics, *Siberian Federal University*, vol. 1, no. 12, pp. 742-744, 2016.
[14] W. Park, and S. Ahn, "Performance Comparison and Detection Analysis in Snort and Suricata Environment", *Wireless Pers Commun*, no. 94, pp. 241-252, 2016, doi: 10.1007/s11277-016-3209-9.
[15] M. Sourour, B. Adel, and A. Tarek, "Network Security Alerts Management Architecture for Signature-Based Intrusions Detection Systems within a NAT Environment", *Journal of Network and Systems Management*, no. 19, pp. 472-495, 2011, doi: 10.1007/s10922-010-9195-4.
[16] Snort and SSL/TLS Inspection, 2017. [Online]. Available: https://www.sans.org/reading-room/whitepapers/detection/snort-ssl-tls-inspection-37735. Accessed on: Dec 10, 2019.
[17] Docker overview, 2020. [Online]. Available: https://docs.docker.com/get-started/overview. Accessed on: Dec 10, 2019.
[18] A. Mouat, Using Docker, *Using Docker: Developing and Deploying Software with Containers*. Newton, USA: O'Reilly Media, 2015.
[19] H. Abbes, T. Louati, and C. Cerin, "Dynamic replication factor model for Linux containers-based cloud systems", *Journal of Supercomputing*, no. 76, pp 7219-7241, 2020, doi: 10.1007/s11227-020-03158-5.
[20] R. Baclit, C. Sicam, P. Membrey, and J. Newbigin, "The Linux Kernel", in *Foundations of CentOS Linux*. California, USA: Apress, 2009, pp. 415-434.
[21] M. Bagherzadeh, N. Kahani, and C.P. Bezemer, "Analyzing a decade of Linux system calls", *Empirical Software Engineering*, no. 23, pp. 1519-1551, 2018, doi: 10.1007/s10664-017-9551-z.
[22] Using eBPF in Kubernetes. [Online]. Available: https://kubernetes.io/blog/2017/12/using-ebpf-in-kubernetes. Accessed on: Dec 10, 2019.
[23] Linux System Call Table. [Online]. Available: https://thevivekpandey.github.io/posts/2017-09-25-linux-system-calls.html. Accessed on: Dec 10, 2019.
[24] S. Adameit, "Modelling Distributed Network Security in a Petri Net- and Agent-Based Approach", in *Lecture Notes in Computer Science*, vol. 6251. Berlin, Germany: Springer, 2010, pp. 209-220, doi: 10.1007/978-3-642-16178-0_20.

The article was received 20.02.2020.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Best Practices for Running Containers and Kubernetes in Production. [Online]. Available: https://www.gartner.com/en/documents/3902966/best-practices-for-running-containers-and-kubernetes- in-. Accessed on: Dec. 14, 2019.
[2] 2019 Container Adoption Survey. [Online]. Available: https://portworx.com/wp-content/uploads/2019/05/2019-container-adoption-survey.pdf. Accessed on: Dec. 14, 2019.
[3] Д. Н Тяжельников, П. А. Токарев, и И. Д. Петров, "Виртуализация рабочего пространства с ускорением 3d-приложений на стороне сервера при помощи Docker", *Проблемы современной науки и образования*, № 14, с. 21-23, 2017.

[4] Infrastructure as Code. [Online]. Available: https://infrastructure-as-code.com/. Accessed on: Dec. 14, 2019.

[5] A. R. Sampaio, J. Rubin, Beschastnikh, N. S. Roca, "Improving microservice-based applications with runtime placement adaptation", *The Journal of Supercomputing*, vol. 10, no. 4, pp. 1-30, 2019, doi: 10.1186/s13174-019-0104-0.

[6] A. Milenkoski, K. R. Jayaram, and S. Kounev, "Benchmarking Intrusion Detection Systems with Adaptive Provisioning of Virtualized Resources", in *Self-Aware Computing Systems*, pp. 633-657, 2017, doi: 10.1007/978-3-319-47474-8_22.

[7] I. Rosenberg, and E. Gudes, "Evading System-Calls Based Intrusion Detection Systems. Network and System Security", in *Proc. International Conference on Network and System Security*, Taipei, Taiwan, 2016, pp. 200-216, doi: 10.1007/978-3-319-46298-1_14.

[8] National Institute of Standards and Technology. (2007, Febr. 20). *NIST SP 800-94, Guide to Intrusion Detection and Prevention Systems*. [Online]. Available: https://csrc.nist.gov/ publications/detail/sp/800-94/final. Accessed on: Dec 10, 2019.

[9] International Organization for Standardization. (2015, Febr. 11). *ISO/IEC 27039, Information technology. Security techniques. Selection, deployment and operation of intrusion detection and prevention systems*. [Online]. Available: https://www.iso.org/standard/56889.html. Accessed on: Dec 10, 2019.

[10] PCI Security Standards Council. (2018, May 01). *Payment Card Industry Data Security Standard.* [Online]. Available: https://ru.pcisecuritystandards.org/_onelink_/pcisecurity/ en2ru/ minisite/en/docs/PCI_DSS_v3_2_RU-RU_Final.pdf. Accessed on: Dec 10, 2019.

[11] M. Aldwairi, A. M. Abu-Dalo, and M. Jarrah, "Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework", *EURASIP Journal on Information Security*, 2017:9, 2017, doi: 10.1186/s13635-017-0062-7.

[12] V. Mishra, V. K. Vijay, and S. Tazi, "Intrusion Detection System with Snort in Cloud Computing: Advanced IDS", in *Proc. of International Conference on ICT for Sustainable Development*, Washington, USA, 2016, pp.457-465.

[13] A. Belova, and D. Borodavkin, "Comparative analysis of intrusion detection systems", Actual problems of aviation and astronautics, *Siberian Federal University*, vol. 1, no. 12, pp. 742-744, 2016.

[14] W. Park, and S. Ahn, "Performance Comparison and Detection Analysis in Snort and Suricata Environment", *Wireless Pers Commun*, no. 94, pp. 241-252, 2016, doi: 10.1007/s11277-016-3209-9.

[15] M. Sourour, B. Adel, and A. Tarek, "Network Security Alerts Management Architecture for Signature-Based Intrusions Detection Systems within a NAT Environment", *Journal of Network and Systems Management*, no. 19, pp. 472-495, 2011, doi: 10.1007/s10922-010-9195-4.

[16] Snort and SSL/TLS Inspection, 2017. [Online]. Available: https://www.sans.org/reading-room/whitepapers/detection/snort-ssl-tls-inspection-37735. Accessed on: Dec 10, 2019.

[17] Docker overview, 2020. [Online]. Available: https://docs.docker.com/get-started/overview. Accessed on: Dec 10, 2019.

[18] A. Mouat, Using Docker, *Using Docker: Developing and Deploying Software with Containers*. Newton, USA: O'Reilly Media, 2015.

[19] H. Abbes, T. Louati, and C. Cerin, "Dynamic replication factor model for Linux containers-based cloud systems", *Journal of Supercomputing*, no. 76, pp 7219-7241, 2020, doi: 10.1007/s11227-020-03158-5.

[20] R. Baclit, C. Sicam, P. Membrey, and J. Newbigin, "The Linux Kernel", in *Foundations of CentOS Linux*. California, USA: Apress, 2009, pp. 415-434.

[21] M. Bagherzadeh, N. Kahani, and C.P. Bezemer, "Analyzing a decade of Linux system calls", *Empirical Software Engineering*, no. 23, pp. 1519-1551, 2018, doi: 10.1007/s10664-017-9551-z.

[22] Using eBPF in Kubernetes. [Online]. Available: https://kubernetes.io/blog/2017/12/using-ebpf-in-kubernetes. Accessed on: Dec 10, 2019.

[23] Linux System Call Table. [Online]. Available: https://thevivekpandey.github.io/posts/2017-09-25-linux-system-calls.html. Accessed on: Dec 10, 2019.
[24] S. Adameit, "Modelling Distributed Network Security in a Petri Net- and Agent-Based Approach", in *Lecture Notes in Computer Science*, vol. 6251. Berlin, Germany: Springer, 2010, pp. 209-220, doi: 10.1007/978-3-642-16178-0_20.

ОЛЕКСІЙ МІСНІК

МОДЕЛЬ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ КОНТЕЙНЕРНИХ СЕРЕДОВИЩ ПРОГРАМНИХ ЗАСТОСУНКІВ

Встановлено призначеність контейнерних середовищ для розроблення, доставляння і експлуатування програмних застосунків різних типів. Серед цих типів найбільш поширене використання як веб, так і мобільних застосунків. Це обумовлено орієнтованістю контейнерних середовищ на їх більш швидке завантаження та встановлення. Використання такого підходу дозволяє розглядати інфраструктуру як код і отримувати пов'язані з нею переваги. Насамперед пришвидшити розроблення програмних застосунків, зокрема, зменшити час між їх написанням і запусканням. Цьому сприяє розгортання контейнерних середовищ на платформах контейнерної віртуалізації, використання утиліт завантаження і керування програмними застосунками. Однак, незважаючи на це використання контейнерних середовищ програмних застосунків на практиці обмежується необхідністю забезпечувати їх безпеку. Насамперед це пов'язано з використанням типових підходів на основі систем виявлення вторгнень. При їх впровадженні поза увагою залишаються особливості використання контейнерних середовищ порівняно з фізичними. Серед них виокремлюються врахування уразливостей і загроз платформ контейнерної віртуалізації, перевіряння процесів контейнерних середовищ з огляду на особливості їхньої архітектури та вхідного потоку навантаження. Для подолання складнощів використання систем виявлення вторгнень запропоновано модель забезпечення безпеки контейнерних середовищ програмних застосунків. За її основу взято ідею використання системних викликів хост системи на прикладі операційної системи Linux. Це пов'язано з тим, що через них відбувається взаємодія програмних застосунків з ядром. Тому виділено користувачів як джерел вірогідних вторгнень у контейнерні середовища. Крім того наведено приклади нетипових команд для аналізування протягом виконання системних викликів. На основі отриманих результатів виокремлено етапи виявлення вторгнень і переходи між ними. Як наслідок, формалізовано даний процес мережею Петрі. Її визначено сукупністю множин етапів, переходів між етапами, відношень між етапами та переходами протягом виявлення вторгнень. Тож на основі запропонованої моделі стало можливим встановлення особливостей забезпечення безпеки контейнерних середовищ програмних застосунків.

Ключові слова: програмний застосунок, контейнерне середовище, безпека, виявлення вторгнень, система виявлення вторгнень, системні виклики.

Misnik Oleksii, postgraduate student, Pukhov institute for modeling in energy engineering of National academy of sciences of Ukraine, Kyiv, Ukraine.
ORCID: 0000-0002-4654-9125.
E-mail: oleksii.misnik@gmail.com.
Міснік Олексій Ігоревич, аспірант, Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова Національної академії наук України, Київ, Україна.