Designing a system call analyser for system calls used inside Linux containers

Marat Nuriev[1,2*], *Rimma* Zaripova[1], *Ramilya* Tazieva[3], *Shamil* Gazetdinov[4], and *Marat* Valiev[5]

[1]Kazan State Power Engineering University, Kazan, Russia
[2]Kazan National Research Technical University named after A. N. Tupolev – KAI, Kazan, Russia
[3]Kazan National Research Technological University, Kazan, Russia
[4]Kazan State Agrarian University, Kazan, Russia
[5]Kazan (Volga region) Federal University, Kazan, Russia

Abstract. This paper examines the development of a system call analyzer used within Linux containers, aiming to ensure security and enhance the performance of containerized applications. Containerization has become a vital aspect of modern software development and operations, enabling efficient isolation of applications and their dependencies. However, selecting a reliable image and analyzing vulnerabilities remain crucial tasks. The focus is on utilizing the ptrace system call and Berkeley Packet Filter technology to monitor and analyze system calls within containers. The developed system call detector interacts with the operating system kernel and the Podman container management tool, ensuring interception and filtration of system calls with minimal impact on container performance. The system's architecture comprises a detector, server, and client components, ensuring modularity, testability, extensibility, and flexible development. The server component processes requests from clients and detectors, manages data, and provides appropriate responses. The client component is a web interface for system interaction. The paper also discusses the functional and non-functional requirements of the system, the implementation tools in Go and JavaScript languages using ReactJS and TypeScript libraries, and the advantages of a multi-layered architecture. The developed system call analyzer contributes to the improved security and performance of containerized applications, as evidenced by testing and system operation results.

1 Introduction

In the modern world, virtualization has become an integral part of software development and operations. One of the most popular approaches to virtualization is containerization, particularly using Linux containers [1,2]. These containers effectively isolate applications and their dependencies, providing maximum flexibility and scalability.

* Corresponding author: marat_nu1@mail.ru

Containers are created from templates called images. Selecting a reliable image is a crucial step in project deployment. The quality and reliability of the image directly impact the application's functionality and security [3,4]. An incorrect image choice can lead to various issues and vulnerabilities that may negatively affect system performance and security.

Analyzing a container is a vital step in the development and operation of applications within a containerized environment. This process provides information about the container's configuration, status, and resources, as well as security checks and optimizations. To ensure container security, a vulnerability analysis is necessary. This includes scanning the container image for known vulnerabilities, checking the container's security settings, and applying necessary patches and updates [5].

However, access to source code for analysis is not always available. In such cases, analyzing a running container by monitoring its system calls is an alternative. This method provides information about the processes [6,7], network interactions, and file system activities within the containers.

The aim of this work is to develop a system call analyzer for use within Linux containers. By utilizing programming languages and operating system tools, it is essential to monitor and analyze system calls within containers to gather information about their interaction with the operating system kernel [8].

Key functional requirements of the system include authorization, adding new system users, restricting user access rights, managing projects, detectors, and scans, detecting system calls within containers, and filtering system calls based on various criteria.

Additionally, there are non-functional requirements [9,10]. The system should minimally impact the performance of Linux containers, support various Linux distributions used for containerization, handle a large volume of system calls in real-time, and possess a user-friendly interface that is easy to configure and use.

2 Ptrace system call and Berkeley Packet Filter technology

Ptrace is a system call available in some Unix-like systems such as Linux, FreeBSD, and Mac OS X, which allows for tracing or debugging a selected process. It provides extensive capabilities for monitoring operations [11,12], including stopping the process, altering its state, reading and writing its memory, registers, and other attributes. By using ptrace, it is possible to intercept system calls, modify their parameters, and return values, but this requires meticulous setup and handling of each kernel call. The prototype of the ptrace function is as follows: `long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);`, where `pid` is the identifier of the process being traced, and `addr` and `data` depend on the `request` [13,14]. Tracing can be initiated in two ways: attaching to an already running process using `PTRACE_ATTACH` or starting it with `PTRACE_TRACEME`. The use of ptrace can be relatively slow due to the significant amount of context switching between user space and kernel space introduced by this method [15].

BPF (Berkeley Packet Filter) is a mechanism built into the Linux kernel designed for packet filtering and analysis at the kernel level. BPF allows embedding user code into the kernel, enabling secure low-level operations in the network stack without root privileges. Developed at the University of Berkeley in 1992, BPF has become a popular tool in Unix-like operating systems [16,17]. While primarily used for processing network packets, implementing network services, and analyzing network activity, BPF can also be utilized for monitoring and analyzing system calls, processes, and other low-level events [18,19].

The key characteristics and principles of BPF include flexibility, high performance, versatility, and extensibility. BPF's flexibility allows user code to be embedded into the operating system kernel [20], enabling real-time operations on packets, analyzing and

filtering them based on various criteria such as destination and source addresses, protocols, ports, and other parameters. BPF's high performance stems from its operation at a very low level in the kernel, allowing for efficient and fast packet processing, monitoring, analyzing, and altering network traffic flow with minimal delay and system load. BPF has numerous applications, including gathering network traffic statistics, packet filtering based on specific rules, debugging network issues, detecting and preventing network attacks, implementing network services, and more [21,22]. BPF can also be used for monitoring system calls and other kernel-level events. BPF's extensibility is supported through mechanisms for loading and executing programs in the kernel, enabling the creation and integration of custom filtering and analysis scripts. Libraries and frameworks, such as eBPF (extended BPF), provide additional capabilities for working with BPF and developing more complex tools and applications. BPF typically performs operations much faster than ptrace due to its ability to safely execute code in the kernel and work directly with network packets and other system data structures [23].

3 System architecture

The detector is a standalone application responsible for launching a container, detecting system calls within it, and transmitting the information to a server. It interacts with the operating system kernel and the Podman container management tool, utilizing eBPF technology to intercept system calls [24,25].

The detector comprises a core and a WebSocket client. The core includes three modules: `container manager`, `pty reader`, and `eBPF` module. The `container manager` handles the lifecycle of containers, including their creation, start, stop, and removal. The `pty reader` processes the output from the containers, while the `eBPF` module intercepts and filters system calls within the containers [26,27].

The architecture of the detector allows it to operate in three modes:

1) Intercepting all system calls.
2) Launching a container and intercepting system calls within the container.
3) Connecting to a server for managing the detector.

This design enables the detector to effectively interact with containers, collecting and analyzing data on system calls, thereby enhancing the security and performance of containerized applications (Figure 1).

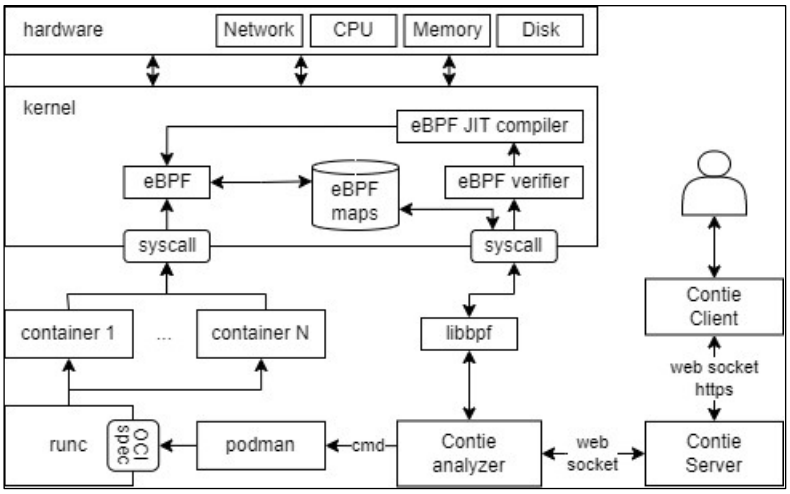

Fig. 1. Diagram of interaction between the detector and the OS.

Figure 2 shows the detector architecture.

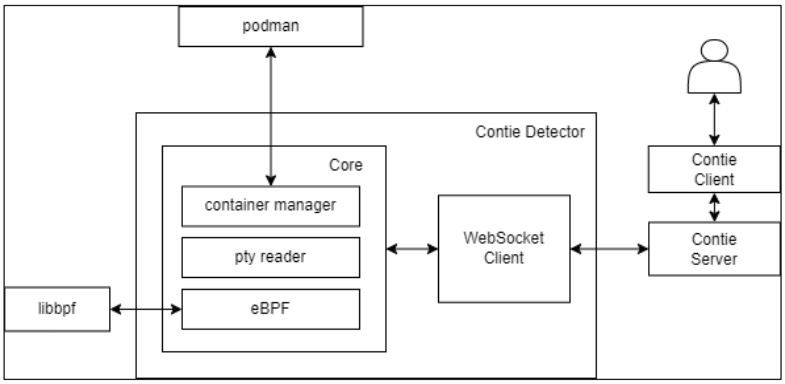

Fig. 2. Detector architecture.

The server component of the application is responsible for processing requests from clients and detectors, managing data, and providing appropriate responses. It interacts with a database for storing and retrieving information. The server architecture is divided into four layers. The model layer represents the entities and domain logic of the application, defining and managing the application's data, including business rules and validations. The repository layer provides an interface for accessing data and interacting with the model layer, encapsulating the implementation details of data storage, such as working with databases or other data sources [28,29]. The service layer offers business logic and operations for the application, utilizing the repository layer for data access and the model layer for data operations. The handler layer handles incoming requests and sends responses, including authentication, authorization, and error handling logic [30].

The interaction between layers works as follows: the handler calls the service, which interacts with the repository for data access. The repository works with the model to perform data operations. The model returns data to the repository, which then returns it to the service. The service sends the data back to the handler, which then sends a response to the client.

This multi-layered architecture has several advantages, including modularity, testability, extensibility, flexibility, scalability, reusability, and reduced complexity. Modularity makes the application more manageable, allowing new features to be added or existing ones modified without rewriting the entire codebase. Testability is enhanced as layers can be tested independently [31,32]. Extensibility allows for the easy addition of new layers or functionalities as the application grows without affecting existing code. Flexibility is ensured by separating business logic from data layers, allowing easy switching of databases or storage solutions without rewriting the application code. Scalability enables independent scaling of different layers, allowing the application to scale according to requirements. Reusability of business logic and data access functions saves development time and effort. Reduced complexity makes the code more understandable and manageable by dividing the application into layers [33,34].

Figure 3 illustrates the server architecture.

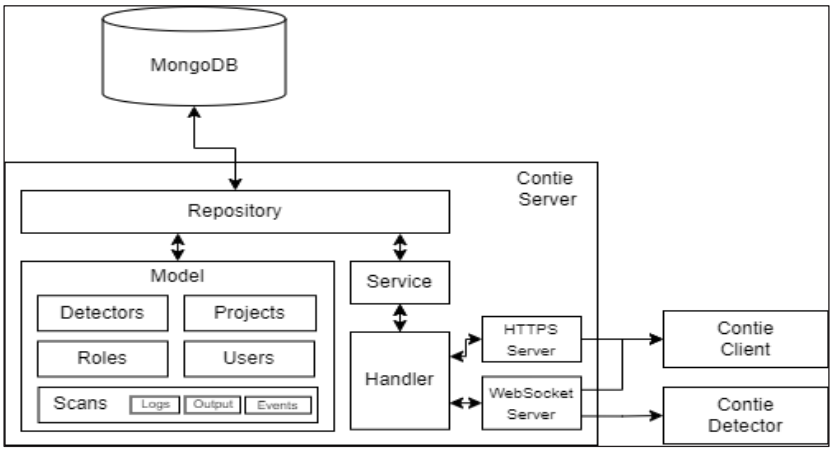

Fig. 3. Server architecture.

The client side of the application will be presented as a web interface consisting of several web pages. These pages provide a user interface for interacting with the service and its functionality. The client-side architecture will be divided into three layers: the router layer, the component layer, and the service layer [35,36].

The router layer manages navigation and routing within the application, responsible for displaying the appropriate component or page based on the URL. The component layer represents a logically isolated part of the user interface, which has its own state, logic, and visual representation. Components can be reused in various parts of the application. The service layer encapsulates business logic and operations that are not tied to a specific component, and is used for data access, performing network requests, or providing other auxiliary functions for the components [37,38].

The interaction between layers works as follows: the router determines which components to display based on the URL. Components interact with services to perform operations and access data. Services provide the necessary data and functionality for the components to display and interact with the user.

This multi-layered architecture allows for the separation of different aspects of the application, making it more modular, testable, and extensible. Such a structure ensures flexibility and convenience in developing and maintaining the client side of the application, allowing for easy addition of new features and components, as well as the maintenance and updating of existing ones [39,40].

Figure 4 illustrates the client architecture.

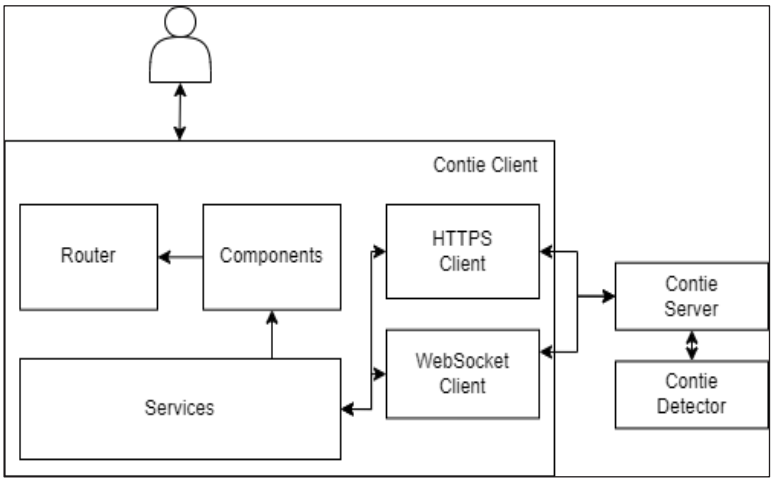

Fig. 4. Client architecture.

4 Implementation toolkit

The Go programming language was chosen for developing the detector and server components. Go is a compiled, statically-typed programming language developed by Google, known for its high performance, scalability, and security [41,42]. Key features of Go include concurrency, built-in support for parallelism and synchronization, flexible typing, a rich standard library, portability, and simplicity. Concurrency supports parallel programming with lightweight threads, making it ideal for multicore systems. Safety is ensured by automatic garbage collection and built-in support for parallelism, reducing the risks associated with multithreaded programming [43,44]. Flexible typing allows for abstracting functionality and increasing code reuse. The standard library provides functionality for networking, data processing, parallel programming, and more. Go's portability lies in its cross-platform nature and the ability to compile into a single executable file, simplifying application deployment. Go's simplicity is achieved through concise and easy-to-learn syntax [45,46]. Code embedded in the Linux kernel is written in the same programming language as the system – Clang.

For developing the client side of the web system, the ReactJS and TypeScript libraries of the JavaScript programming language were selected. ReactJS is an open-source JavaScript library for building user interfaces, created by Facebook. Introduced in 2013, ReactJS quickly gained popularity due to its performance [47,48], flexibility, and efficient use. Key features of ReactJS include its component-based architecture and virtual DOM. The component-based architecture allows the creation of complex user interfaces from reusable and isolated components. The virtual DOM speeds up updates and rendering of the user interface by updating only the parts that have changed [49,50].

TypeScript is an open-source programming language developed and maintained by Microsoft. It extends standard JavaScript by adding static typing and other features. TypeScript's static typing allows defining data types for variables during code writing, which helps identify and fix errors at an early stage [51,52]. TypeScript is used for creating large, complex, and reliable applications, such as full-featured websites, mobile applications, and server-side systems.

Thus, the chosen tools and programming languages ensure high performance, scalability, and reliability of the developed application [53,54], as well as convenience and flexibility in the development and maintenance of the system [55].

5 Conclusion

This work presents the development of a system call analyzer project used within Linux containers. The primary goal of the project is to ensure the security and enhance the performance of containerized applications by analyzing system calls within containers. The implementation of the analyzer utilizes ptrace and Berkeley Packet Filter technologies, enabling efficient monitoring and filtering of system calls at the operating system kernel level.

As a result of the project, a system was developed that includes a system call detector and a server component for data processing and storage. The detector interacts with the operating system kernel and the Podman container management tool, using eBPF technology to intercept system calls. The server component manages data and interacts with clients through a web interface.

The advantages of the system's multi-layered architecture include modularity, testability, extensibility, flexibility, and scalability. This design allows for easy addition of new features and components, as well as maintenance and updates of existing ones.

Thus, the developed system call analyzer provides efficient data collection and analysis, contributing to the improved security and performance of containerized applications.

References

1. L. Quan, Z. Wang, X. Liu, 2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI), Beijing, China, 1-3 (2016)
2. K. Kumar, et al., 15th International Conference on Materials Processing and Characterization (ICMPC 2023), E3S Web of Conferences **430**, 01200 (2023)
3. M. Nuriev, M. Lapteva, E3S Web of Conferences **541**, 02003 (2024)
4. G. R. Mingaleeva, M. F. Nabiullina, D. N. Pham, 2023 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 233-238 (2023)
5. Y. Smirnov, A. Kalyashina, R. Zaripova, International Russian Automation Conference (RusAutoCon), 913-917 (2022)
6. K. M. Vafaeva, N. Duklan, C. Mohan, Y. Kumar, S. Ledalla, International Conference on Recent Trends in Biomedical Sciences (RTBS-2023), BIO Web of Conferences **86**, 01112 (2024)
7. K. M. Vafaeva, R. Zegait, Research on Engineering Structures and Materials **10(2)**, 559-621 (2024)
8. S. Lyasheva, R. Safina, M. Shleymovich, 2023 International Conference on Industrial Engineering, Applications and Manufacturing, 797-802 (2023)
9. K. M. Vafaeva, M. Dhyani, P. Acharya, K. Parik, S. Ledalla, International Conference on Recent Trends in Biomedical Sciences (RTBS-2023), BIO Web of Conferences **86**, 01111 (2024)
10. R. M. Shakirzyanov, A. A. Shakirzyanova, 2021 International Russian Automation Conference (RusAutoCon), 714-718 (2021)
11. Y. I. Soluyanov, A. I. Fedotov, D. Y. Soluyanov, A. R. Akhmetshin, IOP Conference Series: Materials Science and Engineering **860(1)**, 012026 (2020)
12. A. Kalyashina, Y. Smirnov, R. Zaripova, Finance, Economics, and Industry for Sustainable Development. ECOOP 1987. Springer Proceedings in Business and Economics. Springer, 609-618 (2024)

13. L. V. Plotnikova, R. R. Giniyatov, S. Y. Sitnikov, M. A. Fedorov, R. S. Zaripova, IOP Conference Series: Earth and Environmental Science **288**, 012069 (2019)
14. M. Tyurina, A. Porunov, A. Nikitin, R. Zaripova, G. Khamatgaleeva, Lecture Notes in Mechanical Engineering, 391-402 (2022)
15. M. V. Pavlov, et al., International Conference on "Advanced Materials for Green Chemistry and Sustainable Environment" (AMGSE-2024), E3S Web of Conferences **511**, 01036 (2024)
16. Z. Gizatullin, R. Gizatullin, M. Nuriev, 2024 International Russian Smart Industry Conference (SmartIndustryCon), Sochi, Russian Federation, 356-360 (2024)
17. M. Nuriev, A. Kalyashina, Y. Smirnov, G. Gumerova, G. Gadzhieva, E3S Web of Conferences **515**, 04008 (2024)
18. R. F. Gibadullin, N. S. Marushkai, 2021 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 404-409 (2021)
19. K. M. Vafaeva, et al., 15th International Conference on Materials Processing and Characterization (ICMPC 2023), E3S Web of Conferences **430**, 01191 (2023)
20. R. F. Gibadullin, I. S. Vershinin, R. Sh. Minyazev, 2017 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 1-6 (2017)
21. B. R. Reddivari, et al., Cogent Engineering **11 (1)**, 2343586 (2024)
22. K. M. Vafaeva, et al., International Conference on "Advanced Materials for Green Chemistry and Sustainable Environment" (AMGSE-2024), E3S Web of Conferences **511**, 01008 (2024)
23. R. F. Gibadullin, G. A. Baimukhametova, M. Yu. Perukhin, 2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 1-7 (2019)
24. G. R. Rakhmatullina, E. A. Pankova, O. V. Fukina, M. Khayytov, L. V. Chapaeva, Journal of Physics: Conference Series **2270(1)**, 012056 (2022)
25. V. A. Gerasimov, M. G. Nuriev, D. A. Gashigullin, 2022 International Russian Automation Conference (RusAutoCon), 75-79 (2022)
26. Z. M. Gizatullin, R. M. Gizatullin, M. G. Nuriev, 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 120-123 (2020)
27. D. F. Karpov, et al., International Conference on "Advanced Materials for Green Chemistry and Sustainable Environment" (AMGSE-2024), E3S Web of Conferences **511**, 01010 (2024)
28. E. Gracheva, M. Toshkhodzhaeva, O. Rahimov, S. Dadabaev, D. Mirkhalikova, S. Ilyashenko, V. Frolov, International Journal of Technology **11**, 8 (2020)
29. M. Shakirzyanov, R. Gibadullin, M. Nuriyev, E3S Web of Conferences **419**, 02029 (2023)
30. T. Petrov, A. Safin, E3S Web of Conferences **178**, 01016 (2020)
31. J. Yoqubjonov, R. Gibadullin, M. Nuriev, E3S Web of Conferences **431**, 07011 (2023)
32. I. Viktorov, R. Gibadullin, E3S Web of Conferences **431**, 05012 (2023)
33. R. F. Gibadullin, I. S. Vershinin, M. M. Volkova, 2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon), 1-7 (2020)
34. R. F. Gibadullin, M. Yu. Perukhin, B. I. Mullayanov, 2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon), 1-6 (2020)

35. S. N. Cherny, R. F. Gibadullin, 2022 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 965-970 (2022)
36. V. A. Raikhlin, I. S. Vershinin, R. F. Gibadullin, Journal of Physics: Conference Series **2096**, 012160 (2021)
37. R. F. Gibadullin, I. S. Vershinin, R. Sh. Minyazev, 2018 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 1-6 (2018)
38. K. M. Vafaeva, et al., International Conference on "Advanced Materials for Green Chemistry and Sustainable Environment" (AMGSE-2024), E3S Web of Conferences 511, 01037 (2024)
39. V. A. Raikhlin, R. F. Gibadullin, I. S. Vershinin, Lobachevskii Journal of Mathematics **43(2)**, 455-462 (2022)
40. G. A. Ovseenko, R. S. Kashaev, O. V. Kozelkov, T. K. Filimonova, T. S. Evdokimova, A. M. Mardanova, 5th International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE) **5**, 1-5 (2023)
41. R. Zaripova, A. Nikitin, Y. Hadiullina, E. Pokaninova, M. Kuznetsov, E3S Web of Conferences **288**, 01072 (2021)
42. V. Brigida, S. Mishulina, G. Stas, Sustainable Development of Mountain Territories **12(1)**, 18–25 (2020)
43. Z. M. Gizatullin, M. S. Shkinderov, R. R. Mubarakov, Proceedings of the 2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering, 1350-1353 (2022)
44. M. Nuriev, R. Zaripova, O. Yanova, I. Koshkina, A. Chupaev, E3S Web of Conferences **531**, 03022 (2024)
45. A. G. Ilyin, A. S. Mahdi Khafaga, V. Yunusova, 2021 Systems of Signals Generating and Processing in the Field of on Board Communications, 1-4 (2021)
46. R. Zaripova, M. Kuznetsov, V. Kosulin, M. Perukhin, M. Nuriev, E3S Web of Conferences **531**, 03014 (2024)
47. E. Kozlov, R. Gibadullin, E3S Web of Conferences **474**, 02031 (2024)
48. R. M. Petrova, E. Gracheva, 2023 5th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA), Lipetsk, Russian Federation, 1049-1055 (2023)
49. G. Uteyev, R. F. Gibadullin, 2024 International Russian Smart Industry Conference (SmartIndustryCon), Sochi, Russian Federation, 350-355 (2024)
50. N. A. Sabirov, R. F. Gibadullin, 2024 International Russian Smart Industry Conference (SmartIndustryCon), Sochi, Russian Federation, 344-349 (2024)
51. V.S. Brigida, et al., Metallurgist **67**, 398–408 (2023)
52. L. Ma, et al., Resources, Conservation & Recycling Advances **23**, 200224 (2024)
53. V. Romanovski, et al., Physical and Chemical Aspects of the Study of Clusters Nanostructures and Nanomaterials **12**, 293-309 (2020)
54. C.F. Glover, et al., CORROSION **80 (7)**, 755–769 (2024)
55. Ho Lun Chan, et al., Journal of The Electrochemical Society 171, 081501 (2024)