Development and Implementation of Syzkaller Dashboard Alternative for Continuous Linux Kernel Fuzzing

Alexey Panov[1,2], ORCID: 0000-0002-0046-0766
Vladislav Nikolaev[1,3], ORCID: 0009-0009-3257-1311
[1]PJSC Astra Group, 26 Varshavskoe Sh., Moscow, 117105, Russian Federation
[2]Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS), 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation
[3]Moscow Polytechnic University (Moscow Polytech), 38 Bolshaya Semyonovskaya Str., Moscow, 107023, Russian Federation

***Abstract*—The paper focuses on Linux kernel fuzzing and explores potential methods for integrating recent advancements in this field. It evaluates existing software solutions and emphasizes the importance of Continuous Integration (CI) to streamline the fuzzing process. The paper proposes the development of a new web application. This application will include all necessary functions to maximize the use of fuzzing tools under the conditions of economic sanctions and closed software development.**

***Keywords*—kernel fuzzing, syzkaller, continuous integration, web application**

I. Introduction

Incorporating fuzz testing, particularly Linux kernel fuzzing, into the Continuous Integration (CI) pipeline for operating system development is a significant step towards fortifying security against contemporary software threats. The importance of fuzzing the Linux kernel cannot be overstated, as evidenced by the substantial number of high and critical severity vulnerabilities (CVEs) [1]-[5] identified through the use of tools like syzkaller [6]. These findings emphasize the importance of vulnerabilities within the kernel, which pose significant threats to system security. While this approach has been shown to be effective in detecting vulnerabilities, cybersecurity experts face various technical, economic, and political challenges that hinder the smooth integration of fuzz testing into development workflows.

Integrating fuzzing into the CI pipeline can be a challenging task due to the need to incorporate this testing method into established and intricate development processes. This requires automating testing procedures, setting up infrastructure for running fuzzing operations, and automating outcome handling [7]. The integration is further complicated by economic and political factors, particularly the challenges of managing closed development environments or deploying open-source fuzzing tools that depend on proprietary cloud services. The latter may be particularly challenging due to economic sanctions affecting Russian citizens and companies.

However, the value of fuzzing tools like syzkaller is underscored by their proven performance and outcomes, despite the integration challenges. Over six years, syzkaller has identified over 6339 bugs within the Linux kernel, with 5280 subsequently fixed. Nevertheless, the process of discovering these bugs is often protracted, taking on average more than 405 days to detect a single bug [8]. This delay highlights the complexities and the extended timelines that can precede the identification of vulnerabilities by such tools.

The aim of this work is to create an application that is appropriate for the Russian market. The application is intended to optimize fuzz testing capabilities and simplify the integration of the kernel fuzzing process into the CI pipeline for operating system development and its modules within companies that develop domestic operating systems.

II. Contemporary Challenges in Linux Kernel Fuzzing in Russia

Syzkaller stands out as one of the leading and most actively developed tools for Linux kernel fuzzing. However, the full utilization of this system is unavailable not only by the reliance on Google Cloud services [10], which are not accessible to users in Russia, but also by internal corporate policies that restrict using foreign cloud services.

The Technological Center for the Study of Linux Kernel Security, established by ISP RAS, outlines a methodology for using this tool with limited functionality [11]. The usage scenarios include:

- Simple fuzzing scheme. Utilizing standalone setups with different configurations.
- Fuzzing farm scheme. Forming a group of setups with identical configurations and organized information exchange.

Syzkaller comprises the following key services:

- syz-ci, which integrates the kernel fuzzing process into CI, allows for the centralized collection of fuzzing artifacts, such as logs, and bugs in syzkaller's operation. It also automates the process of updating syz-manager's executables when new commits appear in the syzkaller repository. Moreover, this service rebuilds the kernel under test to keep it up to date with the target branch in the kernel repository.
- syz-manager, which manages the virtual machines where fuzzing is conducted.
- syz-hub, which enables the connection of multiple syz-manager instances into a cluster for data exchange. This service helps to minimize the corpus and to determine the necessity of reproducing each crash.

- dashboard, which provides users with information about discovered crashes, organized into bug pages. It also offers statistics on all syz-manager instances, as well as access to fuzzing artifacts, such as build configurations, logs of virtual machines, and compiled images. Furthermore, this service manages task formation for syz-ci and offers an email-based bug-reporting system.

From the entire list of services, the methodology suggests using only two: syz-hub and syz-manager.

The integration of the various approaches presented in the methodology of the Technological Center into the kernel testing process does not permit the full utilisation of the capabilities of Syzkaller. Moreover, it does not facilitate the organisation of the continuous fuzzing process, as the key services are not applied: syz-ci and dashboard. In order for syz-ci to be launched, a working dashboard is necessary.

To compare the schemes described in the Technological Center (TC) methodology with the methodology described in the syzkaller documentation, the organization of service interaction is depicted in the Fig. 1.

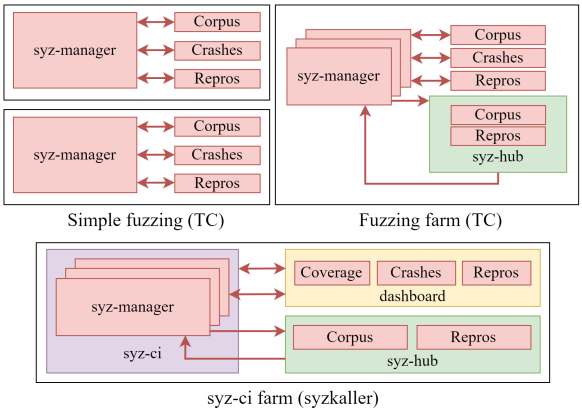

Fig. 1. Syzkaller usage scenarios comparison

The process of the operating system kernel fuzz-testing within the continuous integration cycle can be divided into 5 key stages that form a pipeline:

- Source code changes.
- Building the kernel.
- Starting the fuzzing instance.
- Collection of bugs and fuzzing artifacts.
- Feedback on fuzzing results.

The continuous kernel fuzzing pipeline can be represented as a scheme, as shown in Fig. 2, where the steps that can be automated thanks to syz-ci are highlighted in blue, and the steps that can be automated thanks to dashboard are highlighted in red.

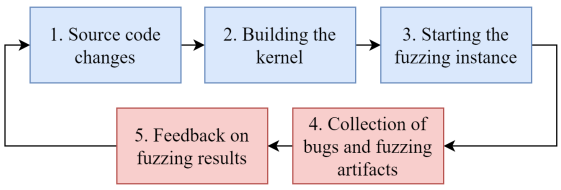

Fig. 2. Continuous kernel fuzzing pipeline

The implementation of the dashboard alternative service will permit the introduction of syz-ci into the Technological Center methodology, which has already been formed. This will allow for the automation of stages that are currently unavailable to Russian developers. Furthermore, it will permit the organization of continuous fuzzing of operating system kernels.

III. Fuzzing OS Kernel with External Kernel Modules

An operating system may include proprietary developments from the company, such as external kernel modules and drivers [12]. Developments can include both user-space utilities and Loadable Kernel Modules (LKM). Therefore, the structure of the operating system kernel development repositories may appear as follows:

- A repository for the Linux kernel source code.
- A repository for the external loadable kernel module (LKM) source code.

Syzkaller is capable of supporting fuzzing and collecting coverage on external kernel modules. However, in order for this functionality to be utilized, a manually built unstripped module for the target kernel must be present in the rootfs image. Additionally, the module must be pre-loaded at a fixed address by the init process. Consequently, the use of syz-ci in the context of continuous fuzzing is not possible.

The requirement for a single syz-manager instance in syz-ci configuration to monitor changes in only one repository, which should contain both the kernel code and the external kernel module code, hinders the setup of a continuous fuzzing process. Furthermore, integrating LKM into the kernel is necessary to address the issue of correctly collecting coverage encountered by syzkaller when testing dynamic modules.

As in [13], "Loadable kernel modules (LKM) are a blessing for the system administrator, but a nightmare for an incident responder. LKMs were initially designed to provide dynamic functionality by altering a running kernel without rebooting", but this solution has brought complexities in crash analysis and coverage collection during fuzzing.

To address this issue, specific kernel patches are required to enable the compilation of the kernel with the external kernel module included.

Fig. 3 shows three repositories:

- Repository with the Linux source code.
- Repository with kernel module source code.
- Repository with the outcome of merging the first two repositories with patches.

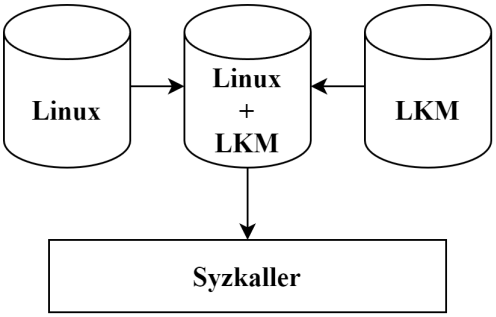

Fig. 3. Typical project scheme for the continuous fuzzing

Consequently, the service to be developed must integrate the LKM code into the Linux kernel code based on the results of the CI job of building the kernel or LKM.

Furthermore, it is possible to implement such a service within a dashboard alternative, which will significantly simplify the implementation of continuous kernel fuzzing within the CI pipeline.

IV. Review of Existing Solutions

The time required to discover vulnerabilities and the time needed to achieve maximum code coverage with tests both increase proportionally to the size of the codebase of the project being tested. Due to the significant amount of time required for fuzzing, integrating these tools into the conventional "linear" CI/CD pipeline poses challenges. Additionally, it's important to consider that the fuzzing process itself demands substantial computational resources. In the context of automated kernel fuzzing to maximize efficiency, this could involve monitoring the status of dozens, sometimes even more, virtual machines within which multiple test suites of system call chains may be executed in parallel. Consequently, researchers in this field have found a solution to these challenges by creating managed clusters of virtual machines. On these clusters, fuzzers are run, and the results of their work are sent to reporting systems, thereby optimizing the fuzzing process within the constraints of available resources and workflow integration.

Prominent examples of platforms for implementing continuous fuzzing include ClusterFuzz [14], OSS-Fuzz [15], and syzbot [16]. ClusterFuzz stands out as a scalable infrastructure designed for fuzzing user-space projects. This project is utilized by Google to test all its products and serves as a fuzzing component in the OSS-Fuzz project. ClusterFuzz offers a lot of features for integrating fuzzing into the software development process. Notably, its high scalability is commendable, as it can be deployed on a cluster of up to 100,000 virtual machines. The system's ability to deduplicate crashes, minimize test cases, identify the commit that introduced a regression in the repository, and automatically log discovered crashes in bug tracking systems like Jira or Monorail also merits mention.

ClusterFuzz operates on the Google Cloud Platform and leverages several services:

- Compute Engine (not strictly necessary, fuzzing bots can run anywhere).
- App Engine. The App Engine instance provides a web interface to access crashes, statistics and for scheduling cron jobs.
- Cloud Storage.
- Cloud Datastore.
- Cloud Pub/Sub.
- BigQuery.
- Stackdriver Logging and Monitoring.

It's possible to run ClusterFuzz locally using Google Cloud emulators for those not requiring these dependencies, though it's worth noting that features dependent on BigQuery and Stackdriver might be limited due to the absence of emulator support.

OSS-Fuzz represents a production instance of ClusterFuzz, enriched by the OSS-Fuzz repository's content, including build scripts and project.yaml files with contact information. Tailored for the testing of open-source projects, OSS-Fuzz delivers a robust infrastructure aimed at verifying their reliability and security through automated fuzzing.

Both OSS-Fuzz and ClusterFuzz are engineered to streamline the fuzz testing process for user-space programs, aiding in the detection of vulnerabilities and enhancing software dependability via comprehensive automated testing. Nevertheless, it's critical to understand that they are not adapted for Linux kernel testing. Their design and methodologies are specifically intended for user-space applications, rendering them inappropriate for kernel-level examination.

The syzbot platform is built around the syzkaller fuzzer. It continuously scans the main branches of the Linux kernel repository, rebuilding the kernel when new commits are detected. It also restarts fuzzing instances and automatically reports any discovered bugs to mailing lists, in addition to offering a monitoring dashboard that displays the status of all bugs. Syzbot is also noted for its scalability, as it can utilize either Google Compute Engine (a virtual machine in Google Cloud) or local Qemu virtual machines for fuzzing operations. Furthermore, syzbot enables the testing of patches for bugs that have a generated reproducer, enhancing the efficiency of addressing and correcting vulnerabilities within the kernel.

All of the aforementioned systems were developed under the guidance of Google, and therefore rely on Google Cloud services for their operation:

- Google Compute Engine provides virtual machines for fuzzing operations.
- Google Cloud Storage serves as an object storage solution for archiving coverage files, corpora, build files, or executables.
- Google App Engine offers a platform for delivering monitoring dashboards and managing the fuzzing process.

Therefore, it can be concluded that it is impossible to use such systems under sanctions pressure or when developing software in a closed environment. TriforceAFL is an example of clusterization without using paid services. However, it coordinates multiple fuzzing instances through the creation of shared folders and writing of bash scripts, which is not as automated as the aforementioned systems. Although the automation is imperfect, it permits the fuzzing of LKM modules.

A comparison of fuzzing farms presented in Table 1 reveals that Syzbot is the most promising system in the field of Linux kernel fuzzing. However, it lacks the functionality offered by farms in the field of user-space program fuzzing, such as support for bug-tracking systems like Jira. Additionally, it does not allow the user to fully deploy a local instance of the system, as some of the services within syzbot have a direct dependence on cloud services.

TABLE I. Fuzzing platforms comparison

Feature	Cluster Fuzz	OSS-Fuzz	syzbot	Triforce AFL
Linux kernel fuzzing support	-	-	+	+
LKM fuzzing support	-	-	+/-	+
Running a local instance	+/-	-	+/-	+

Feature	Cluster Fuzz	OSS-Fuzz	syzbot	Triforce AFL
Linux kernel fuzzing support	-	-	+	+
Dependence on cloud services	+	+	+	-
GUI	+	+	+	-
Bug-tracking systems support	+	+	+	-
Alerting	+	+	+	-
Active development	+	+	+	-
Ways of interaction between systems components	HTTP, RPC	HTTP, RPC	HTTP, RPC, SSH	File system, SSH

V. Proposed Solution

This paper proposes the development of a web-based application to replace the current service integrated into syzkaller—the dashboard. This web application will allow users to access information on active fuzz testing setups, detected crashes, and generated reproducers.

In order to fully utilize syzkaller and ensure continuous fuzz testing, it is proposed that additional functionality be added to the web application under development. This functionality will involve integrating the LKM code with the operating system kernel, saving the results to a specified repository, as defined in the syz-ci configuration file.

The proposed solution offers several advantages:

- There is no need for modifications in the syzkaller's source code, which avoids conflicts during synchronization between the original repository and the local version, thereby simplifying maintenance by the company's staff.
- Isolating the functionality into a separate service facilitates the automation of launching continuous fuzzing processes. Additionally, it does not restrict the addition of extra functionalities. For instance, the service could be integrated into a reporting system, allowing an administrator to integrate modules into the kernel and rebuild images for all syz-manager instances with a simple button press in the web interface. This approach not only enhances operational efficiency but also opens the door for future expansions and customizations tailored to specific organizational needs.

The backend component of a web application consists of three logical components:

- Dashboard API (Application Programming Interface) for syz-managers and syz-ci.
- Frontend API.
- Integration service.

The dashboard API is designed to emulate the functionality of the original dashboard, with minor changes. This component allows forming entities to store information about running syz-managers and their configurations, as well as to obtain information about detected crashes. Consequently, the key functionality of syz-ci is employed, and the dependency on the dashboard service is eliminated. Instead, a full-fledged analog will be provided, which can be freely customized to meet the requirements of a particular company. Furthermore, this approach will eliminate the need to manually track the information provided by each of dozens of instances, as it will introduce bug report aggregation.

The frontend API is essential for enabling interactions between the web client and the server. This component implements the logic required for displaying information to the user, including the generation of pages with bugs and the tracking of fuzzing instances states.

The integration service is an API that facilitates the receipt of integration requests and a set of asynchronous tasks for interacting with the file system and Git. These tasks include the following:

- Moving directories and files.
- Applying patches.
- Cloning repositories.
- Generating and sending commits.

The interaction between the components of the proposed continuous fuzzing solution in the Fig. 4 is as follows:

1. The software developer makes changes to the source code of the tracked projects (Linux and LKM) in the version control system.
2. The Linux and LKM CI build jobs are started in the development VCS respectively.
3. If successful, a notification is sent to the Integration Service.
4. The integration service starts the process of integrating the two repositories into a 3rd repository (fuzzing repository).
5. The syz-ci pulls changes and starts the kernel build for fuzzing, runs tests and starts fuzzing.
6. The syz-ci sends build information results, syz-managers' errors, coverage reports and corpora to the dashboard API.
7. The syz-manager sends detected crashes and work statistics to the dashboard API.
8. The security analyst then reviews the crashes found by syzkaller and creates issues in the bug tracking system.

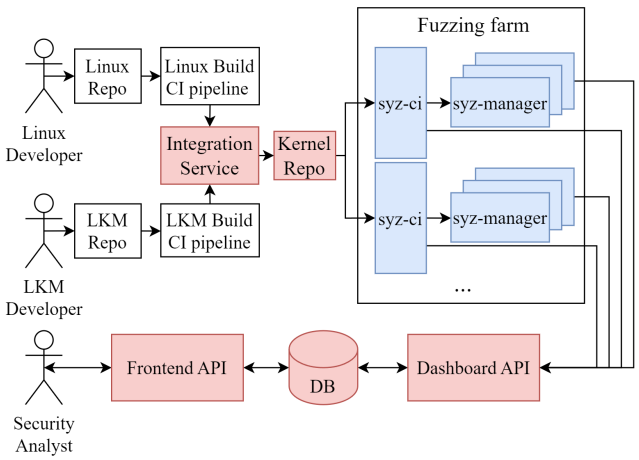

Fig. 4. Components interaction

VI. Implementation

To construct the application, an architecture as depicted in the Fig. 5 was chosen. The components related to the

frontend aspect of the application are highlighted in blue, backend components are in purple, databases and object storage are marked in green, and syzkaller components are in yellow.

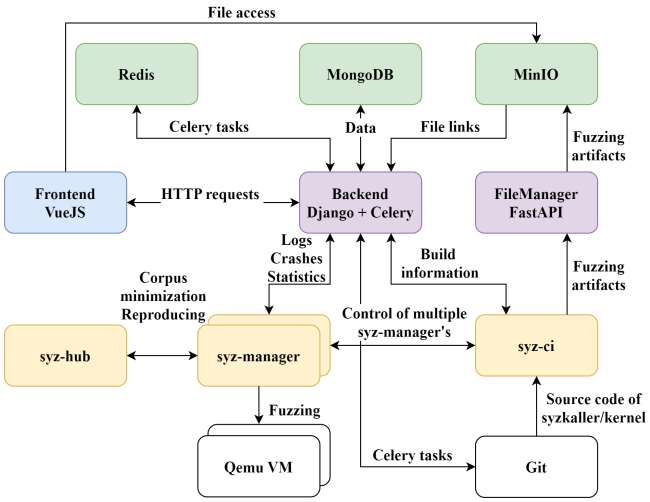

Fig. 5. Proposed system architecture

It is assumed that syzkaller components will be deployed on high-performance servers with a large number of virtual machines running syz-manager instances. The fuzzing process itself will be launched by the systemd service that runs a Docker container with syz-ci.

Docker Compose enables the execution of all frontend and backend services of a web application in Docker containers. This approach simplifies the development and management of services, while also providing opportunities for scaling and rapid deployment of the entire system.

Consequently, the infrastructure scheme of the entire project will take the form depicted in Fig. 6.

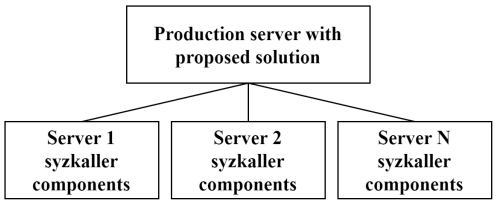

Fig. 6. Infrastructure scheme of the entire project

A. *Backend Implementation*

The server-side of the application is divided into several services: the backend and the FileManager. The backend is developed using Python [17] with the Django REST Framework [18]. It comprises three types of components:

- Models, which describe the objects in the database.
- Views, which define the REST API handlers.
- Serializers, which detail how to represent data received from the client or the database.

Data storage is managed with MongoDB [19], utilizing the Djongo library [20] to facilitate the application's connection to the database and the description of models.

To handle tasks such as tracking new commits and integrating two repositories into one, the Celery task queue [21] is employed, which is integrated into a Django application.

The task of collecting fuzzing artifacts, such as fuzzing corpus and coverage reports, is handled by the FileManager service, written in Python using the FastAPI framework [22]. This service accepts HTTP PUT requests from syz-ci, stores files in object storage, and stores links to files in the database.

B. *Frontend Implementation*

The website is designed as a responsive Single Page Application (SPA) [23]. A Single Page Application is a website consisting of one HTML document that dynamically updates and does not require the entire page to reload during use.

The client-side was developed using the JavaScript [24] programming language, with the VueJS framework [25] and Vuetify [26]. Vuetify, in combination with VueJS, facilitates the rapid development of the web application's user interface. The vue-router library is responsible for organizing routing to match application requests with specific interface components. Meanwhile, the Axios library [27] is used to form HTTP requests to the web application's backend.

This architecture not only supports the seamless integration and functionality of the proposed web application but also provides a scalable and efficient platform for continuous fuzzing and vulnerability management. By leveraging modern frameworks and libraries, the system ensures a robust infrastructure for enhancing software security through automated fuzzing processes.

VII. Conclusion and Future Work

In this paper, we propose a web-based application aimed at enhancing the accessibility of continuous kernel fuzz testing. By leveraging the capabilities of syzkaller and addressing the limitations of existing solutions, our proposed system offers a scalable platform for continuous fuzz testing that is specifically designed to meet the needs of developers operating under various constraints, including economic sanctions and closed development environments.

The proposed solution enables the full implementation of syz-ci within the continuous integration pipeline. It provides a convenient interface for displaying fuzzing information and facilitates the integration of LKMs into the Linux kernel source code for further continuous testing.

Our solution offers two key advantages: maintenance simplicity and service isolation. The combination of contemporary development libraries and frameworks, robust backend implementation, and a user-friendly frontend design provides the foundation for a platform that streamlines the continuous kernel fuzzing process.

As we improve our solution, we plan to extend its capabilities by adding support for bug tracking systems such as Jira to streamline vulnerability management.

In addition to improvements to facilitate integration of fuzzing into CI, we also intend to incorporate state-of-the-art approaches into the platform with the objective of enhancing the efficiency of the fuzzing process itself, such as KernelGPT [28]. The authors explore the use of Large Language Models (LLMs) to automatically infer syzkaller specifications for improved kernel fuzzing. KernelGPT's approach to iteratively infer and refine specifications presents a compelling avenue for automating and improving syscall sequence generation, thereby enhancing coverage and uncovering previously unknown bugs.

In addition, SyzDirect [29] introduces a novel method for directed greybox fuzzing (DGF) tailored to the Linux kernel. SyzDirect improves bug reproduction and patch testing efficiency by using scalable static analysis to identify critical system calls and argument conditions to reach target code locations. Its ability to quickly reproduce more bugs and reach more target patches than conventional methods makes it a promising direction for future efforts in directed fuzzing for patch verification.

Finally, the integration of approaches to automate fuzzing with promising enhancements to the process itself will allow us to achieve better results in exploring vulnerabilities in the Linux kernel. By identifying these vulnerabilities at an earlier stage of the development process, we can reduce the likelihood of exploitation.

References

[1] "CVE-2023-2124" [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-2124

[2] "CVE-2022-48502" [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-48502

[3] "CVE-2022-0850" [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-0850

[4] "CVE-2022-1055" [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-1055

[5] "CVE-2016-9555" [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2016-9555

[6] "Coverage-guided kernel fuzzing with syzkaller" [Online]. Available: https://lwn.net/Articles/677764

[7] H. Shi, et al., "Industry practice of coverage-guided enterprise linux kernel fuzzing", In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 986-995.

[8] J. Bursey, A. Amiri Sani, and Z. Qian, "SyzRetrospector: A Large-Scale Retrospective Study of Syzbot", arXiv preprint arXiv:2401.11642, Jan 2024.

[9] "Mezhdunarodnyi proekt po razrabotke yadra Linux [International Linux Kernel Development Project]" [Online]. Available: https://portal.linuxtesting.ru/about-linux-kernel.html (in Russian)

[10] "Google Cloud Documentation" [Online]. Available: https://cloud.google.com/docs

[11] "Metodika fazzing-testirovaniya yadra [Kernel fuzzing method]" [Online]. Available: https://portal.linuxtesting.ru/LVCFuzzingMethod.html (in Russian)

[12] "Nvidia Unix Driver Archive" [Online]. Available: https://www.nvidia.com/en-us/drivers/unix

[13] K. Jones "Loadable kernel modules", login: The Magazine of USENIX and SAGE, 2001, vol. 26, no. 7.

[14] "ClusterFuzz - Scalable fuzzing infrastructure." [Online]. Available: https://google.github.io/clusterfuzz

[15] "OSS-Fuzz - continuous fuzzing for open source software." [Online]. Available: https://google.github.io/oss-fuzz

[16] "syzbot" [Online]. Available: https://syzkaller.appspot.com/upstream

[17] "Python" [Online]. Available: https://www.python.org/

[18] "Django REST framework" [Online]. Available: https://www.django-rest-framework.org/

[19] "MongoDB Documentation" [Online]. Available: https://www.mongodb.com/docs/

[20] "Django and MongoDB database connector" [Online]. Available: https://www.djongomapper.com/

[21] "Celery - Distributed Task Queue" [Online]. Available: https://docs.celeryq.dev/en/stable/index.html

[22] "FastAPI framework" [Online]. Available: https://fastapi.tiangolo.com/

[23] V. Gavrilă, L. Băjenaru, C. Dobre. "Modern single page application architecture: a case study", Studies in Informatics and Control, 2019, vol. 28, no. 2, pp. 231-238.

[24] "JavaScript" [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[25] "Vue.js" [Online]. Available: https://vuejs.org/

[26] "Vuetify - Vue Component Framework" [Online]. Available: https://vuetifyjs.com/en/

[27] "Axios HTTP Client" [Online]. Available: https://axios-http.com/docs/intro

[28] C. Yang, Z. Zhao, L. Zhang, "KernelGPT: Enhanced Kernel Fuzzing via Large Language Models", arXiv preprint arXiv:2401.00563, Dec 2023.

[29] X. Tan, Y. Zhang, J. Lu, X. Xiong, Z. Liu, M. Yang. "Syzdirect: Directed greybox fuzzing for linux kernel" In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (pp. 1630-1644), Nov 2023.