Automated Vulnerability Scanner for the Cyber Resilience Act

Sandro Falter*, Gerald Brunkh†, Max Wess‡ and Sebastian Fischer§

Dept. of Computer Science and Mathematics
Ostbayerische Technische Hochschule Regensburg
Regensburg, Germany
Email:
sandro.falter@st.oth-regensburg.de*, gerald.brukh@st.oth-regensburg.de†,
max.wess@st.oth-regensburg.de‡, sebastian.fischer@oth-regensburg.de§

***Abstract*—This paper explores the mitigation of the compliance burdens faced by manufacturers of digital products under the Cyber Resilience Act. After providing a concise overview of the Cyber Resilience Act and pinpointing pivotal areas where tool-based interventions could reduce the regulatory strain on manufacturers, we introduce two prototypes: a digital checklist for product classification and a prototype to streamline the analysis and monitoring of the security state of software along the software development life cycle. As the second prototype is based on Static Application Software Testing and Software Component Analysis, we validate the approach through benchmark tests. While Static Application Software Testing tools show promise in identifying vulnerabilities, additional tests are needed for full compliance with the Cyber Resilience Act. In general, the prototypes serve as an entry point for identifying possible automation potential to alleviate the compliance burdens of manufacturers.**

***Keywords*—cra; cyber resilience act; vulnerability scanner; reporting; iot; cloud**

I. Introduction

A few years ago, Richard Clarke summarized the importance of cyber security with the following pointed statement: “If you spend more on coffee than on IT security, you will be hacked ” [1]. The statement seems exaggerated at first, but appears in a new light, especially in the area of cyber security, when you consider a report issued by the World Economic Forum [2].

The report shows that the rapid advancements in the area of IoT have created a lack of standards and regulations. Governments, individual organizations, and households rely on IoT devices to power their infrastructure. However, the lack of standardization and cybersecurity considerations left them vulnerable to attacks that stifle future adoption of IoT [2, p.5].

The study reports that 82% of respondents have low confidence that connected devices and related technologies are protected against the unethical and irresponsible use of the technology [2, p.8]. Furthermore, 73% of respondents have low confidence that connected devices are secured and users are protected against attacks [2, p.14]. Forecasts show that this problem will continue to worsen in the coming years. According to the report, global cybercrime is expected to grow 15% per year over the next five years, raising yearly induced costs of cybercrime to $ 10.5 trillion a year [2, p.16].

To address these issues, the EU proposed a new regulation called the Cyber Resilience Act (CRA) on 15th September 2022 [3]. The regulation focuses on products with digital elements and introduces new mandatory cybersecurity requirements for hardware and software products throughout the whole lifecycle.

1) **Risk Assessment:** Emphasis on Security by Design, products shall be delivered without known vulnerabilities. Regular tests and security reviews need to be performed.
2) **Documentation:** To prove conformity, the CRA also requires reports about the tests carried out to show the absence of vulnerabilities. In addition, a Software Bill of Materials (SBOM) must be provided listing the included third-party libraries, packages, and dependencies.
3) **Vulnerability Reporting:** Known vulnerabilities must be reported within 24 hours to the European Union Agency for Cybersecurity (ENISA).

In a nutshell, the CRA requires the monitoring of the security state of products with digital elements along the whole life cycle, including the development phase, release phase, and operation phase. These compliance guidelines lead to additional overhead for manufacturers of digital products.

To address these problems, this paper attempts to provide an overview of possible solutions and approaches that could reduce the overhead of companies with digital products. The paper focuses on solutions that can be implemented in practice and benefit companies during operations thus answering the following research questions:

1) **RQ 1:** Which areas of complying with the CRA could be covered by tool support?
2) **RQ 2:** How could prototypes look that implement this tool support?
3) **RQ 3:** How could the performance of the tools be measured?

The paper is structured as follows: in Section 2, the related work is given. In Section 3, a CRA compliance checklist is

 ISBN: 978-1-68558-156-5

presented as one part of the paper. In Section 4, the second part, a vulnerability scanner, and the related performance analysis is shown. Subsequently, in Section 5, we give a short discussion and in the end in Section 6, the conclusion is given.

II. Related Work

The introduction of the CRA in 2024 is expected to boost research activities in cybersecurity and compliance. However, in academia, there are limited research efforts for providing tool-based assistance to help manufacturers with aligning with the CRA. This necessitates a more comprehensive approach to understand the current research landscape. Past reports by ENISA, such as [4] or [5], offer guidelines applicable to CRA compliance, but lack clear recommendations. While numerous studies focus on compliance checking of software processes, such as Ardila et al.'s literature review [6] and Barati et al.'s framework for GDPR compliance verification [7], they do not directly address cybersecurity and compliance as required by the CRA. Caris et al. [8] developed a framework and a web-based application to aid small and medium-sized companies in achieving cyber resilience. In the non-academic sector, various compliance management software like ZenGRC from RiskOptics [9] and Cloudsmith's artifact management platform [10] assist in compliance checking, with Cloudsmith also addressing CRA compliance through vulnerability scanning and software bill of materials creation. However, academic response to CRA challenges remains limited, contrasting with early efforts in the commercial sector toward CRA compliance.

III. CRA Compliance Checklist

To further combat the limited academic response to CRA challenges, a web-based application tailored to streamline the compliance process for individual products was developed. This application is designed with a focus on user accessibility, presenting CRA requirements in a format that is straightforward and digestible. Users are empowered to utilize this digital tool to navigate through requirements related to CRA adherence.

The initial function of the application is to discern products that are not subject to CRA oversight, such as those governed by specific regulatory exceptions or those that are open-source in nature. Subsequently, for products that fall within the purview of CRA regulations, the tool systematically categorizes them into Class I, Class II, or a default category. It provides a comprehensive list of product types laid out by the CRA to assist in accurate classification. Upon selection of the most appropriate product type by the user, the application provides detailed information regarding the product's classification and the subsequent obligatory criteria that must be satisfied for CRA compliance.

In the third phase, the user engages with the application by addressing specific compliance-related requirements laid out by the CRA. These questions are structured to elicit responses that not only affirm compliance but also allow for commentary, thereby creating a record that can enhance the understanding of CRA compliance or link to relevant analyses or subject matter. Moreover, the tool dispenses practical recommendations and best practices to aid manufacturers in meeting each requirement with greater ease. This feature is particularly beneficial in simplifying the CRA's implications for product manufacturers and streamlining the compliance process. The insights and documentation generated through this interactive tool are invaluable, serving as a robust foundation for the compilation of the manufacturer's EU declaration of conformity.

Figure 1 illustrates the user journey for a microprocessor manufacturer, serving as a visual guide to the various options and functionalities available within the prototype. When a Product is considered excluded from the CRA, users are presented with the option to proceed with the analysis of the current product. This feature is designed to accommodate the possibility if the product may be exempt at present, the requirements outlined by the CRA could become applicable in the context of future product developments. Beyond the standard compliance verification pathway, the prototype offers the flexibility to navigate across different sets of questions. This adaptability ensures a personalized experience, enabling users to tailor the compliance process to meet their specific needs and circumstances.

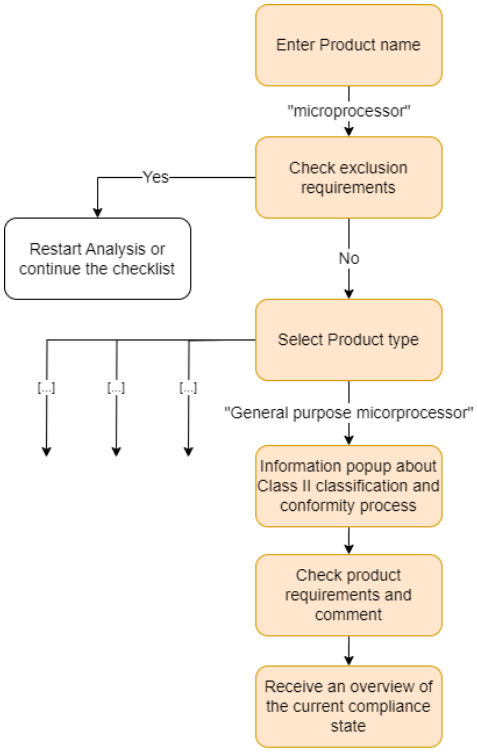

Fig. 1. User Story Chart for a Microprocessor Manufacturer. This diagram outlines the step-by-step process within the compliance tool, from entering the product name to receiving a comprehensive overview of the product's compliance with the CRA

To enhance the understanding of the compliance status, users are provided with a comprehensive summary upon completion of the checklist. This summary offers a clear and concise overview of the compliance state, facilitating a better grasp of the overall situation. For the purposes of this paper,

 ISBN: 978-1-68558-156-5

this overview is depicted in Figure 2, which serves to visually represent the final compliance assessment.

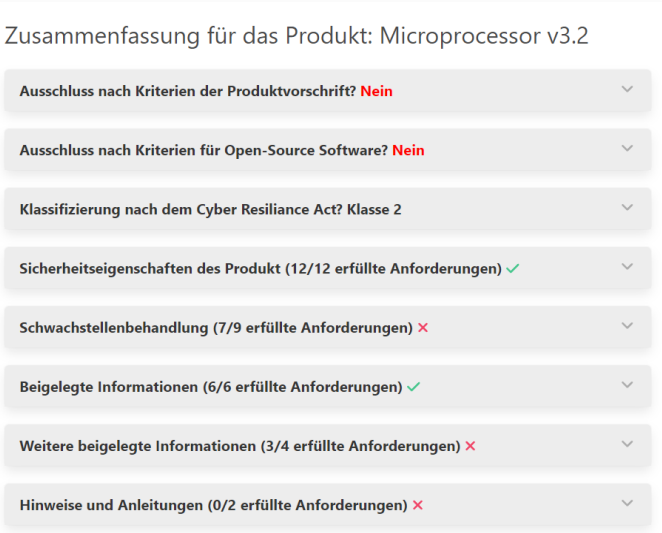

Fig. 2. Example overview of Compliance Assessment for Microprocessor v3.2. This summary displays e.g. the product's exemption status, classification under the CRA, and the fulfillment of security features

The following requirements ensure that the Compliance Checklist operates smoothly, provides a user-friendly experience, and effectively guides users through the CRA compliance process:

A. Functional Requirements

1) **FR 1:** As a user, I want the tool to check if my product falls within the regulations of the CRA
2) **FR 2:** As a user, I want the tool to categorize my product into the associated class, according to the rules of the CRA
3) **FR 3:** As a user, I want to get a list of all necessary requirements for my specific product
4) **FR 4:** As a user, I want to be able to document the compliance of my product with each requirement
5) **FR 5:** As a user, I want to save my documentation and be able to edit them again later
6) **FR 6:** As a user, I want to export my documentation as a PDF document

B. Quality/Non-Functional Requirements

1) **NFR 1:** User-friendly structure
2) **NFR 2:** Understandability even for users with little technical knowledge

IV. Vulnerabilty Scanner

The second tool was developed in response to the CRA's requirement to analyze and monitor the security state of software with digital elements along the whole software lifecycle. In the first step, we analyzed the structure of current DevSecOps approaches to adapt similar solutions to the field of IoT and the CRA. Conventional DevSecOps pipelines implement security practices from the earliest stages of planning and design and also cover operational stages after shipping the product [11, p.4]. This includes the following stages:

1) Manual Code Reviews
2) Software Component Analysis (SCA)
3) Static Application Security Testing (SAST)
4) Penetration Tests
5) Unit, Integration, System and Acceptance Tests
6) Dynamic Application Security Testing (DAST)
7) Configuration Management Testing
8) Security Monitoring

The prototype of the software focuses on a holistic approach and is intended to map central elements of this DevSecOps pipeline that can be automated. We identified the Software Component Analysis and Static Application Security Testing as most suitable for automation, as they can be generalized for different code repositories. For example, Penetration Tests and Unit Tests are challenging to automate and generalize as they are highly specific for a respective code base. Based on this analysis we defined the following requirements for the prototype.

A. Functional Requirements

1) **FR 1:** As a user, I am able to scan for vulnerabilities in a given project to assist with my self-assessment
2) **FR 2:** As a user, I want to have a visualization of all detected vulnerabilities to get a better understanding of the current security status of the project
3) **FR 3:** As a user, I want to be able to schedule scans to get regular reporting on the current security state of the project
4) **FR 4:** As a user, I would like to generate an SBOM report that provides an overview of all the components of a given repository
5) **FR 5:** As a user, I would like to get an overview of all included vulnerabilities in the dependencies listed in the SBOM
6) **FR 6:** As a user, I would like to scan repositories in the following languages as they are mainly used in IoT Development: C, C++, Python

B. Quality/Non-Functional Requirements

1) **NFR 1:** Usability and simplicity of operation
2) **NFR 2:** Flexibility of Deployment of the Application
3) **NFR 3:** Integration into the Development process
4) **NFR 4:** Flexible Expandability of the Application

In a nutshell, the application aims to assist users with self-assessment for CRA compliance by scanning repositories for vulnerabilities and providing reports. It includes features, such as visualization of detected vulnerabilities, scheduling scans for regular reporting, generating SBOM reports, analyzing dependencies, and supporting languages commonly used in IoT development (C, C++, Python). Non-functional requirements prioritize usability, flexibility of deployment, integration into development processes, and flexible expandability.

 ISBN: 978-1-68558-156-5

C. Architecture of the Tool

1) Design Decisions: The design decisions are based on the previously defined requirements. The application is segregated into a server-client architecture, where the user interface constitutes the frontend service developed using Vue.js [12], while the backend encompasses multiple services responsible for repository analysis and resource management. The Hypertext Transfer Protocol (HTTP) is employed to facilitate communication between the front and backend. A web-based frontend enables the flexible distribution of the prototype to all conventional operating systems, like Linux, Windows and macOS. Certain functionalities within the application are augmented through the integration of third-party software. This third-party software is conceptualized as an additional microservices and is accessible via a terminal interface. Third-party applications are Git [13] for repository management, Syft [14] to create the initial textfiles listing all dependencies of the analyzed code repository, and additional Static Application Security Testing Tools like Semgrep [15], Flawfinder [16]and CppCheck [17]. Initially, it was planned to use Horusec [18] as well. However, technical problems during the evaluation complicate a representative comparison with the other scanners. These tools are utilized for vulnerability detection through code analysis. These tools are integrated into the backend Docker container [19] to streamline the scanning process. Third-party services are used to retrieve personal information regarding the target repositories to be analyzed and to provide additional data on the Software Component Analysis. These services are the GitHub API [20] and the sonatype OSS Index [21].

2) Structure of the Prototype: Figure 3 shows the Deployment View of the created prototype. The system is distributed into multiple nodes, which are the user's computer, the docker execution environment, and the two external nodes Git Hub and the sonatype OSS Index API. The client can simply access the application via a web browser, no additional dependencies are required to use the application. The NGINX reverse proxy will serve the front end, which also includes a web server. In addition, the NGINX reverse proxy is responsible for the routing between the backend and frontend docker containers. After receiving the website, the client can provide his Git Hub Account credentials to a formula in the frontend and the website will then proceed to fetch an overview of the user's repositories from the GitHub API. After this, the client can select one or multiple target repositories for a vulnerability analysis or SBOM generation. After starting the vulnerability scan, the backend will fetch the respective target repositories from GitHub and then proceed with scanning the repositories with the SAST vulnerability scanners. For the SBOM generation, the backend needs to include additional vulnerability information which is provided by the OSS index database. The backend therefore fetches the necessary data from the OSS index for the respective packages and dependencies included in the target repositories.

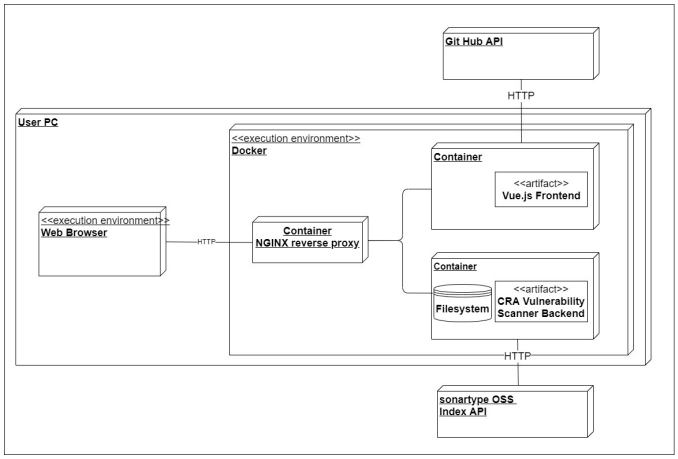

Fig. 3. Deployment overview of the prototype. The Vulnerability Scanner consists of a Vue.js frontend and a Python backend. Docker Compose is used to orchestrate the containers.

D. Performance of the Vulnerability Scanner

The performance of the vulnerability detection tool is largely dependent on how well the respective SAST tools work. There are multiple approaches to evaluate the performance of SAST tools. Most approaches are based on benchmarks. The benchmarks contain one or more repositories for which the number of vulnerabilities is known. The individual scanners are then used to analyze the benchmark repositories and results are compared. We based this evaluation on the approach of [22] and selected the Juliet Test Suite for C/C++ 1.3.0 [23] and Wireshark 1.8.0 [24] as benchmark repositories. Both repositories are published and maintained by the National Institute of Standards and Technology (NIST). The Juliet Test Suite Benchmark consists of 64099 synthetic test cases. The Wireshark Benchmark, on the other hand, is modeled on a real project. The vulnerabilities are therefore not synthetically generated but were discovered through vulnerability analysis in the project. We suspected that individual SAST developers might adapt their tool to the synthetic benchmarks, which is why the additional Wireshark data set is intended to improve the quality and significance of the results.

1) Evaluation Approach: For evaluation the scanners we used the following approach:

(a) *Preprocess Ground Truth Data:* The Benchmarks and their files are preprocessed. All files of the Wireshark Benchmark are used for the evaluation. For the Juliet Test Suite we excluded sophisticated text cases that span across multiple files. The benchmarks contain additional ground truth data, which includes information, such as the line number, type and location of the vulnerability. These datasets are loaded into a database.

(b) *Analyze the Benchmark repositories with the SAST Scanners:* We perform vulnerability detection on each benchmark repository with the following scanners - Semgrep 1.41.0, Flawfinder 2.0.19, and Cppcheck 1.4.0. All the scanners were used in their default configuration. The

 ISBN: 978-1-68558-156-5

TABLE I
RESULTS OF THE JULIET BENCHMARK

Juliet Test Suite for C++ 1.3.0, - 40626 vulnerabillities			
	Flawfinder	**Semgrep**	**CppCheck**
True Poitives	11159	0	3662
False Positives	189617	9556	7191
Precision	5,6%	0%	33,7%
Recall	27,4%	0%	9%

TABLE II
RESULTS OF THE WIRESHARK BENCHMARK

Wireshark 1.8.0 - 767 vulnerabillities			
	Flawfinder	**Semgrep**	**CppCheck**
True Poitives	14	0	0
False Positives	1466	231	55
Precision	1%	0%	0%
Recall	1,8%	0%	0%

results are also loaded into the database to simplify the comparison between results and ground truth data.

(c) *Comparision between Scanner Results and Ground Truth data* The results and ground truth data are available in the database. There are now several ways to compare the results. In this evaluation, we have assumed that we evaluate an exact match on the file path and the line of code as a true positive. This is necessary because all scanners support a different output format and provide different information on the detected vulnerability. The file path and line number are specified across all scanners. If a scanner detects a vulnerability and matches the file location and the line number exactly to on entry in the ground truth dataset then this is considered a true positive.

2) Results: Tables I and II show the results of the scanning and the comparison. The Juliet Test Suite consists of 28 Million lines of code with a total of 40626 vulnerabilities. Each vulnerability points to a specific file location and line in the code. As Table I shows, Flawfinder performed the best out of all scanners. It detected 11159 of the 406226 vulnerabilities correctly. However, it produced over 189617 false positives in the process. This leads to a low precision rate. The precision is defined as the number of true positives compared to the total number of scanner findings. Here just 5,6% of detected Flawfinder vulnerabilities are actual vulnerabilities. The recall rate is better, at least around one-quarter of all vulnerabilities have been detected. The other scanners especially Semgrep performed badly. No true positives have been detected and therefore the recall and precision are at 0%. Cppcheck performed better. 9% of all vulnerabilities were found, but nearly every third of them was a true positive. Therefore developers using Cppcheck face less noise than developers using Flawfinder, however, fewer vulnerabilities are detected in general. All scanners performed worse on the non-synthetic Wireshark Benchmark. Only Flawfinder was able to generate True Positives. But with a Precision rate of 1% users will get 99 false positives for one actual vulnerability.

V. DISCUSSION

In general, the SAST scanners were able to detect some vulnerabilities but not enough to reach full compliance with the Cyber Resilience Act. There are multiple reasons to explain the performance. As observed, the scanners performed better with the synthetic Juliet Benchmark than the non-synthetic Wireshark Benchmark. SAST scanners work by applying a set of fixed rules to a given codebase. The quality of the results depends on the quality of the rule databases which can be accessed by the scanner. The Juliet Benchmark has been specifically created to benchmark SAST tools. Therefore, the vulnerabilities included in the Juliet Benchmark are more in line with the actual rule sets and capabilities of SAST tools. The Wireshark Benchmark is based on actual code which includes some vulnerabilities. The reason for performance differences for example between Semgrep and Flawfinder can be explained by the number of rules available for each scanner. Taking a closer look at the code repositories on GitHub one can conclude that Flawfinder defines 169 rules [25] for pattern matching, and Semgrep defines 13 rules [26]. The quantity does not state anything about the quality of the rule sets but must be considered as a factor when comparing the performance of the scanners. In addition, SAST tools don't analyze the code during runtime, the scanners lack context awareness and therefore can not detect vulnerabilities under more realistic conditions. In addition, some scanners produce many false positives which creates additional noise for developers and makes identifying actual security risks in the code more difficult. Some scanners provide configuration options to reduce the amount of false positives, but this often involves a trade-off of detecting fewer true positives as a result.

VI. CONCLUSION

This study explored new approaches and possible solutions to reduce the compliance overhead of manufacturers of digital products that fall under the Cyber Resilience Act. We provided a quick overview of the Cyber Resilience Act and identified key areas where tool-based assistance could reduce the burden for manufacturers. The first prototype is a digital checklist that helps clients classify their products following the new risk classes introduced by the Cyber Resilience Act. The tool enables the documentation of the compliance process and helps identify action items to meet the compliance criteria of the Cyber Resilience Act. The second prototype represents an initial attempt to streamline the analysis and monitoring of the security state of software along the software development life cycle. To achieve this we identified key testing stages along a DevOps pipeline and identified Static Application Security Testing and Software Component Analysis as two central testing steps that can be automated and improve the security state of a software. Subsequently, a prototype has been developed and tested to validate the approach. The evaluation showed that our approach can be a first foundation to develop a more holistic approach to monitoring the security state of the software. The SAST tools can detect some vulnerabilities, but to achieve full compliance with the Cyber Resilience

 ISBN: 978-1-68558-156-5

Act, additional tests are necessary to identify weaknesses and cybersecurity risks in the code.

REFERENCES

[1] R. Lemos, "Security Guru: Let's Secure the Net," 2024, [Online; accessed: 2024-02-27].
URL https://www.zdnet.com/article/security-guru-lets-secure-the-net/

[2] S. Ahmed, M. Carr, M. Nouh, and J. Merritt, "State of the Connected World," Tech. rep., World Economic Forum, Jan. 2023.

[3] European Commission, "Regulation of the European Parliament and of the Council on horizontal cybersecurity requirements for products with digital elements and amending Regulation (EU) 2019/1020," 2022, [Online; accessed: 2024-02-27].
URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:52022PC0454

[4] C. Skouloudi, A. Malatras, R. Naydenov, and G. Dede, "Guidelines for Securing the Internet of Things," Tech. rep., ENISA, 2020.

[5] ENISA, "Baseline Security Recommendations for IoT in the Context of Critical Information Infrastructures," Tech. rep., European Union Agency For Network And Information Security, Nov. 2017.

[6] J. P. Castellanos Ardila, B. Gallina, and F. Ul Muram, "Compliance Checking of Software Processes: A Systematic Literature Review," Journal of Software: Evolution and Process, 34(5), p. e2440, 2022, ISSN 2047-7481, doi:10.1002/smr.2440.

[7] M. Barati, G. Theodorakopoulos, and O. Rana, "Automating GDPR Compliance Verification for Cloud-hosted Services," in 2020 International Symposium on Networks, Computers and Communications (ISNCC), pp. 1–6, Oct. 2020, doi:10.1109/ISNCC49221.2020.9297309.

[8] J. F. Carías, S. Arrizabalaga, L. Labaka, and J. Hernantes, "Cyber Resilience Self-Assessment Tool (CR-SAT) for SMEs," IEEE Access, 9, pp. 80741–80762, 2021, ISSN 2169-3536, doi:10.1109/ACCESS.2021.3085530.

[9] RiskOptics, "RZenGRC," 2024, [Online; accessed: 2024-02-27].
URL https://reciprocity.com/product/zengrc/

[10] cloudsmith, "cloud native artifact management," 2024, [Online; accessed: 2024-02-27].
URL https://cloudsmith.com/product/cloud-native-artifact-management

[11] F. Lombardi and A. Fanton, "From DevOps to DevSecOps Is Not Enough. CyberDevOps: An Extreme Shifting-Left Architecture to Bring Cybersecurity within Software Security Lifecycle Pipeline," Software Quality Journal, 31(2), pp. 619–654, Jun. 2023, ISSN 1573-1367, doi:10.1007/s11219-023-09619-3.

[12] "Vue.Js," 2024, [Online; accessed: 2024-02-27].
URL https://vuejs.org/

[13] "Git," 2024, [Online; accessed: 2024-02-27].
URL https://git-scm.com/

[14] "Anchore/Syft," Anchore, Inc., Jan. 2024.

[15] "Semgrep — Find Bugs and Enforce Code Standards," 2024, [Online; accessed: 2024-02-27].
URL https://semgrep.dev/

[16] "Flawfinder Home Page," 2024, [Online; accessed: 2024-02-27].
URL https://dwheeler.com/flawfinder/

[17] "Cppcheck - A Tool for Static C/C++ Code Analysis," 2024, [Online; accessed: 2024-02-27].
URL https://cppcheck.sourceforge.io/

[18] "Horusec," 2024, [Online; accessed: 2024-02-27].
URL https://horusec.io/site/

[19] "Docker: Accelerated Container Application Development," May 2022, [Online; accessed: 2024-02-27].
URL https://www.docker.com/

[20] "GitHub: Let's Build from Here," 2024, [Online; accessed: 2024-02-27].
URL https://github.com/

[21] S. Inc, "Sonatype OSS Index," 2024, [Online; accessed: 2024-02-27].
URL https://ossindex.sonatype.org/

[22] C. Gentsch, "Evaluation of Open Source Static Analysis Security Testing (SAST) Tools for C," Technical Report DLR-IB-DW-JE-2020-16, DLR German Aerospace Center, Jan. 2020.

[23] National Institute for Standards and Technology, "Juliet C/C++ 1.3 - NIST Software Assurance Reference Dataset," 2017, [Online; accessed: 2024-02-27].
URL https://samate.nist.gov/SARD/test-suites/112

[24] National Institute for Standards and Technology, "Wireshark 1.8.0 - NIST Software Assurance Reference Dataset," 2014, [Online; accessed: 2024-02-27].
URL https://samate.nist.gov/SARD/test-suites/94

[25] D. A. Wheeler, "Flawfinder/Flawfinder.Py at Master · David-a-Wheeler/Flawfinder · GitHub," https://github.com/david-a-wheeler/flawfinder/blob/master/flawfinder.py.

[26] "Semgrep-Rules/c/Lang/Security at Develop · Semgrep/Semgrep-Rules · GitHub," https://github.com/semgrep/semgrep-rules/tree/develop/c/lang/security.

 ISBN: 978-1-68558-156-5