



Secure Software Development Life Cycle

Security; DevSecOps; DevOps; CI/CD; Software Development; Pipelines; Organizations; Secure SDLC

White paper

Version 1.0, August 2023

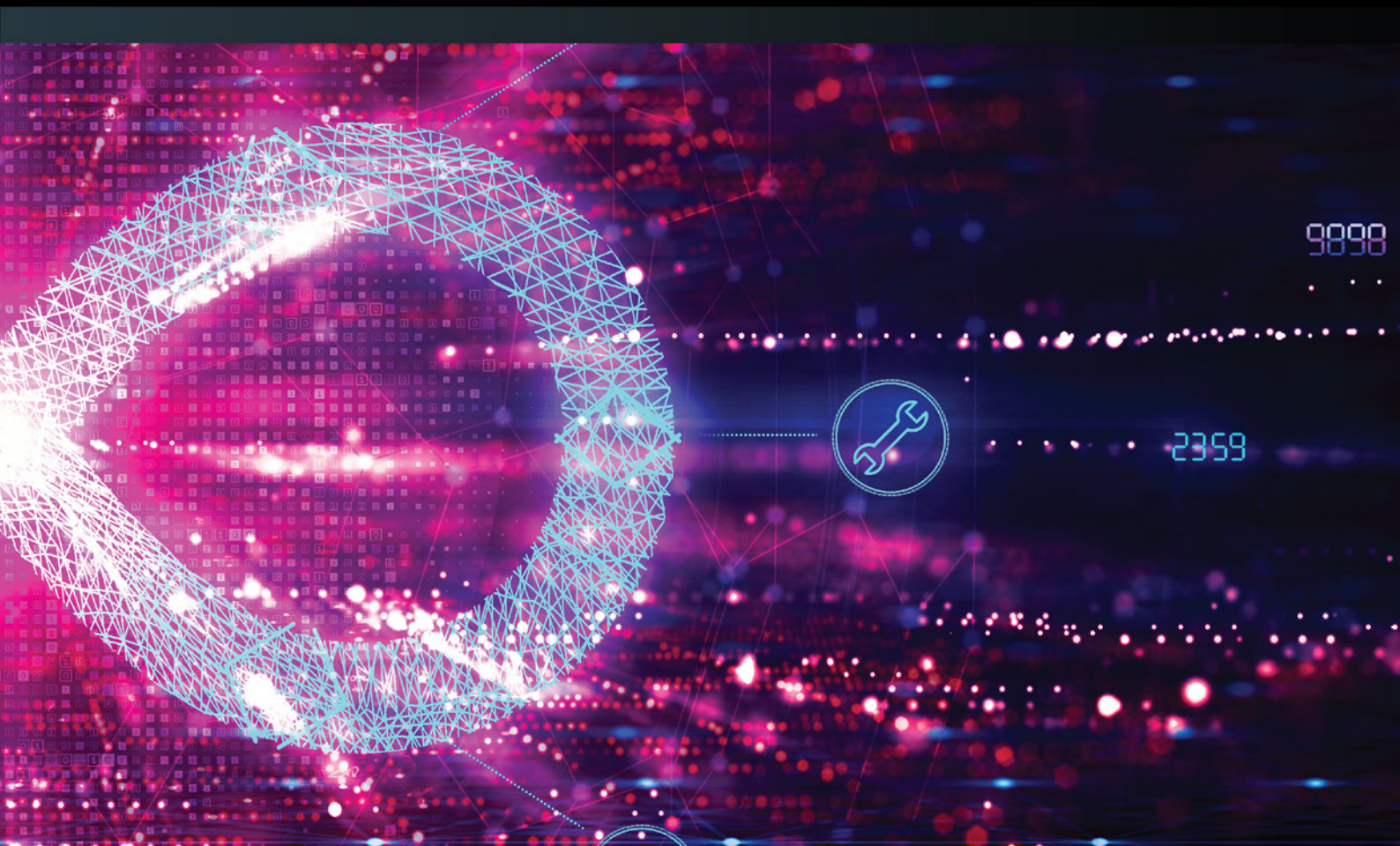
Introduction

For the past years, many organizations across the globe chose to deviate from the classic software development methodology for the sake of adopting a newer and more advantageous one [1]. The development process for software used to be linear and sequential, using the classic waterfall approach. This method was rigid and slow, which made it challenging for businesses to adapt to the shifting consumer needs and market demands. As a result, businesses started implementing more agile methods of software development, such as Scrum and Lean, which emphasized the incremental and iterative delivery of software.



Nonetheless, despite this adoption of agile approaches, there was still a barrier between the development and operations teams. While operations were in charge of ensuring the stability and dependability of the current systems, development focused on delivering new features and capabilities. The distribution of software was delayed by this division, which frequently resulted in disputes, as well as security vulnerabilities and breaches [2].

The DevOps approach was introduced to address these issues, combining both the software development process and the IT operations silos that were formerly separated [3]. The adoption of these new principles and methodologies allowed companies to reach new velocity and agility levels previously unattainable [1]. From a DevOps point of view, automation is key. In this approach, the automation of the software development process is intrinsically connected to the concept of Continuous Integration and Delivery (CI/CD), which refers to the pipeline that stands between the development team and the client, ensuring an automated process of building, testing, and deploying the application. This automation process made developing and releasing software faster and more reliable than ever.



However, despite the numerous advantages of the DevOps approach, the software development life cycle (SDLC) still lacked the integration of security in every phase. On one hand, manual security processes can be a bottleneck for the fast release of software, since they take time and are complex. On the other hand, security is indispensable and when ignored can cost companies huge sums of money, as well as make both the stakeholders and clients lose confidence in the brand.

Modern organizations depend on their Software Supply Chain since it encompasses the development, distribution, and maintenance of the software they use, supporting a variety of activities and processes. The security of the supply chain must be ensured to safeguard against cyber threats and data breaches, which can have serious financial and reputational repercussions for organizations.

Just in 2022, Sonatype revealed a 742% average annual increase in Software Supply Chain attacks over the past 3 years [4]. Furthermore, it was shown in an analysis conducted by IBM that 83% of the organizations studied had had more than one data breach, with the global average cost of a single one rounding up to 4.35 million USD [5]. In that same study, it was also shown that not only did one-fifth of breaches occur as a result of supply chain compromise, but that the average cost of a breach related to supply chain compromise was 2.5% higher (4.46 million USD) than the total average cost of a data breach [5].

More recently, attacks in the supply chain, like the Solar-winds attack [6], and the approval of more strict regulations, like the European GDPR and the NIS2 Directive, and other US federal and state legislation to increase the cybersecurity of American organizations and citizens (e.g., FISMA and CCPA), put tremendous pressure on the need for various industries to ascertain the security of their products and services. They responded by trying to integrate automated security processes into the DevOps approach, complementing the CI/CD pipelines and allowing the shift to a DevSecOps paradigm, giving meaning to the term "Continuous Security". The shift to DevSecOps presents some interesting challenges, such as how to secure both the applications being developed and the CI/CD pipelines themselves. Another important and relevant concern is how to flawlessly integrate these security methods into existing DevOps processes without disturbing their fast pace and responsiveness.

In a study from 2022 that explored the challenges faced while adopting DevSecOps [7], researchers found that industries considered the highest challenging factors were the lack of automated testing tools for security in DevOps and the identification of software defects found in operations monitoring and feeding them back to development, both top-rated by 85% of the industries surveyed.

Industries are increasingly acknowledging the necessity of integrating robust security measures throughout the various phases of the SDLC. This recognition is highlighted by the alarming statistics, the increasing volume of attacks, and the emerging strict legislative frameworks discussed earlier. To this end, there has been a progressive adoption of automated security measures within existing DevOps practices, with the goal of significantly enhancing the security posture of software development processes. However, industries face significant challenges in this transition to DevSecOps, primarily due to the lack of standard tools designed to facilitate this change. The objective of the work presented in this paper is to tackle this exact issue, enabling the transition to DevSecOps with the ultimate aim of bolstering the security of the entire software development process and Software Supply Chain.

DevSecOps-enabled SDLC

The growing significance of software security in the development process has drawn a lot of attention in recent years to the adoption of DevSecOps principles and practices. However, incorporating security measures into the quick and agile development processes poses multiple difficulties. In this section, some of the key security concerns that must be addressed in each phase of the DevSecOps life cycle are identified. **Figure 1** illustrates these phases, integrating the Development (Dev), Security (Sec), and Operations (Ops) aspects into a unified model.

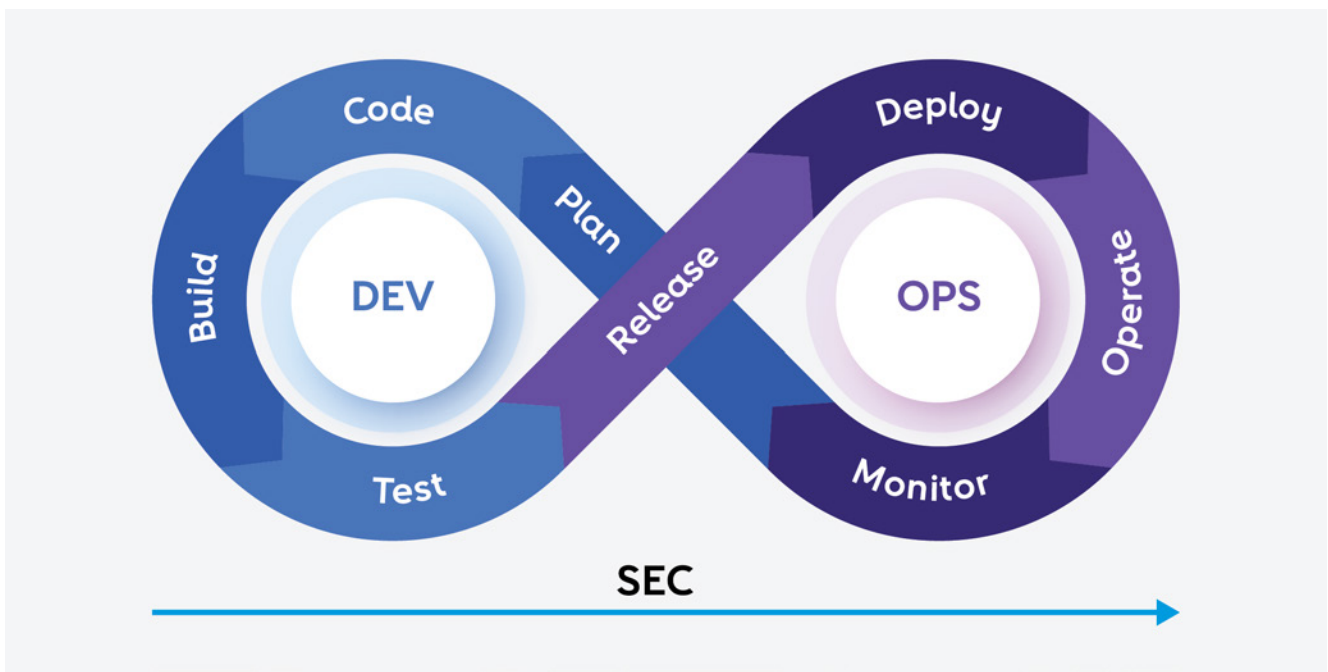


Figure 1 - DevSecOps-enabled SDLC

The Development phases reflect the initial stages, such as planning, coding, building, and testing. The Operations phases represent the latter stages where the application is released, deployed, operated, and monitored. Security traverses and intertwines with all phases, ensuring it is not an afterthought, but rather a constant focus, pervading every aspect of the development and operations processes. This also emphasizes the “shift left” security approach in DevSecOps, where security is considered earlier in the life cycle to identify and address issues as soon as possible.

The feedback loop format of the figure indicates that the DevSecOps process is iterative and cyclical. After each cycle, feedback is taken from monitoring and applied to improve the next iteration, promoting continuous improvement and learning. This ensures that the development, operation, and security aspects of the application are constantly updated and improved, leading to higher quality and more secure software.

Transversal security concerns in DevSecOps include selecting automation tools, access control, knowledge sharing, pipeline security, secrets management, and fostering a security-aware culture. Key practices involve

Infrastructure as Code (IaC), which automates the setup of computing environments and resources, and Configuration as Code (CaC), which ensures consistent configurations across environments. Managing various environments, whether on-premise, cloud, hybrid, single-tenant or multi-tenant, also presents unique security challenges.

In the **planning phase** of DevSecOps, secure code training sharpens developers' security skills. Threat modeling is used to identify potential security threats. Security and privacy risk assessments focus on protecting data integrity and privacy. Change impact analysis predicts the potential effects of system modifications, while abuse case analysis anticipates system misuse. These steps inform the definition of security requirements for the team to follow.

In the **coding phase** of DevSecOps, practices like source code versioning and signed commits ensure traceability and authenticity of code changes. Automated tools aid code reviews for quality assurance, while commit hooks enforce coding standards. Secret scans detect sensitive data, preventing leaks. IDE plugins for Static Application Security Testing (SAST) and Software Composition Analysis (SCA) improve code security. Workstation security policies protect development environments.

The **build phase** of DevSecOps involves employing secure build techniques to create reliable, secure software. It also includes the use of signed dependencies to verify the integrity and authenticity of third-party software components, ensuring a trusted Software Supply Chain.

The **testing phase** of DevSecOps integrates several strategies for robust security. Secrets injection manages sensitive data during testing. SCA, SAST, and secrets scans detect vulnerabilities in code and potential data leaks. Dynamic Application Security Testing (DAST) and Interactive Application Security Testing (IAST) identify security threats in running applications.



The **release phase** is the final checkpoint in DevSecOps before deployment, making it crucial for rigorous security testing. Techniques like SCA, SAST, and secret scans are extensively used to guarantee the security of the code. The package's digital signature/encryption, coupled with secure transfer, ensure its safety throughout the process. A Software Bill of Materials (SBOM) provides full transparency of the package's contents. Extensive DAST and IAST are performed, preferably in Quality Assurance (QA) environments that closely mimic the production environment, to avoid disrupting operations while ensuring the utmost security of the application.

During the **deployment phase** of DevSecOps, penetration testing (pentesting) is conducted to identify potential vulnerabilities that could be exploited in a real-world scenario. Runtime protection is implemented to safeguard applications while they are running. Additionally, host security measures are enforced, and vulnerability scans are performed to ensure the deployment environment's security.

The **operation phase** of DevSecOps involves strategies for continued security and maintenance. Log collection provides valuable insight into application behavior and potential security incidents. Runtime Application Self-Protection (RASP) and Web Application Firewalls (WAF) actively safeguard the application. Cloud Native Application Protection Platforms (CNAPP) offer holistic protection in cloud environments. Bug bounty programs incentivize the identification of vulnerabilities, while regular patching ensures the mitigation of identified security risks.

In the **monitoring phase** of DevSecOps, CNAPP provide continuous oversight of applications in cloud environments. Security Information and Event Management (SIEM) systems collect and analyze security-related data for threat detection. Extended Detection and Response (XDR) integrates various security tools for a comprehensive defense strategy. Continuous vulnerability monitoring identifies and addresses emerging security threats, ensuring the ongoing security of the application.



Altice Labs software development process

This section explores some key aspects of the SDLC as practiced by Altice Labs. Understanding these practices will provide a foundation for suggesting improvements, based on the DevSecOps methodology, to enhance not just the security embedded within the SDLC but also within the Software Supply Chain. The integration of DevSecOps can strengthen Altice Labs' ability to combat security risks and ensure the development of secure, efficient, and robust software solutions.



Agile practices

Altice Labs originally operated under the Waterfall model of software development. This model, while well-structured, lacks the flexibility and iterative nature found in Agile methodologies. Recognizing these limitations, Altice Labs has integrated Agile practices, specifically Scrum, into their development processes and also adopted continuous integration. Looking towards the future, Altice Labs anticipates further optimizing its software development and delivery processes through the adoption of the DevSecOps methodology.



Standardized solutions

Altice Labs has been transitioning from proprietary solutions and custom on-premise components, such as registries, towards more widely-accepted and standardized solutions like GitHub Registries and Google Cloud Artifact Registries. However, the transition is challenging due to the tight coupling and integration between custom components. Moving towards these standardized solutions is a strategic move that not only aligns with industry best practices but also addresses the challenges of maintaining and enhancing custom solutions, particularly in terms of security.



Knowledge centralization

Currently, Altice Labs maintains both internal and external instances of Jira and Confluence, leading to increased costs and potential inefficiencies. To streamline this, plans are underway to consolidate into single instances of Cloud Jira and Confluence. This centralization aims to reduce costs, enhance collaboration, ensure data consistency, and improve overall operational efficiency.



Workstation security

Current security practices at Altice Labs involve access control measures and two-factor authentication (2FA), with varying privileges for users based on job roles. Antivirus software, firewall policies, and full disk encryption are also used. While SAST plugins for secure code development and SCA plugins for dependency security scanning are already in use, they have not been standardized or uniformly adopted across the entire organization. The goal moving forward is to establish a more consistent and widespread application of these tools.



Source code management

Altice Labs has been transitioning from Apache Subversion (SVN) to Git, specifically GitHub, for source code management. Git provides multiple appealing features, such as distributed version control, a robust security model, and commit signing to ensure code authenticity. GitHub provides some additional security features that will be used, such as protected branches and pull request reviews, making sure the requirements for code review, approval, and status checks are met.



Security tests

Altice Labs plans to standardize the use of security testing tools across projects, integrating system tests after each build, and utilizing SCA, SAST, DAST, and IAST. There is also a special focus on QA-intensive security testing, as this environment closely mimics the production environment and does not cause service disruption.



Continuous integration

Currently, CI is used for code integration, automated building, and testing, predominantly managed by Jenkins. In the future, Altice Labs plans to standardize the use of CI and transition from Jenkins to GitHub Actions for managing CI processes.

Tools analysis

Several security tools on the market can help organizations and individuals safeguard their systems and data [8]. These tools identify possible vulnerabilities in systems and applications, thus helping to identify and stop security attacks and breaches. There are different kinds of security analysis tools, including SAST [9], SCA [10], and DAST [11], among others. Each of these types of tools has a distinct purpose, and they can be used in conjunction to build a robust CI/CD pipeline [8].

As previously mentioned, Altice Labs is progressively adopting GitHub Actions. So, this CI/CD platform will be used to construct the CI/CD pipeline and multiple security tools will be integrated with it to detect and prevent security threats. The purpose of this pipeline is to provide continuous security throughout the development life cycle, from code commits to deployment. Organizations may ensure that their code is more secure and vulnerabilities are found and resolved as early as feasible by introducing automated security checks at every stage of the process.

Free security analysis tools are very helpful for people and companies who might not have the dedicated funds to invest in pricey security solutions. It is also vital to keep in mind that while these tools can be helpful, they might not be as feature-rich as their paid counterparts, even though they offer numerous capabilities.

Various free security analysis tools were compared in this section to build a secure CI/CD pipeline that is both efficient and economical. Furthermore, these tools were chosen with deep consideration for Altice Labs' current tech stack to address any existing security gaps within its systems. This way, security measures can be implemented without disrupting the established workflows, aligning with the principles of DevSecOps, which advocates for the seamless integration of security into the development and operations processes.

Types of security analysis

As previously mentioned, there are various types of security analysis, three of which are the focus of this work: SAST, DAST, and SCA. **Table 1** is presented below, highlighting and comparing these analyses key characteristics.

	Static Analysis	Dynamic Analysis	Composition Analysis
Type	White-Box	Black-Box	Gray-Box
Requirements	Source Code	Running Application	Metadata / Source Code
SDLC phases	Code; Test; Release	Test; Release; Deploy	Code; Test; Release
Targets	Source Code, IaC, and CaC	Running applications and Infrastructure	Open-source SW, 3 rd -party libraries, and dependencies
Findings	Bugs, vulnerabilities, secrets, and code smells	Runtime vulnerabilities and security weaknesses	Known vulnerabilities, licensing issues, and outdated libraries

Table 1 - Types of security analysis

In this table, the type of analysis denotes the level of system knowledge during testing (white, black, or gray box). The requirements field lists what is analyzed by the tool. The SDLC phases pinpoint when the tool can be implemented in the development life cycle. The targets indicate to which scopes the analysis is applied. Lastly, the findings refer to the type of security issues the analysis can detect.

This comparative understanding aids in choosing the right tools for integration into the CI/CD pipeline, ensuring robust and cost-effective security measures. These tools can be used in combination for a comprehensive security analysis.

Tool comparison criteria

When researching free security tools, a combination of academic literature and web-based resources were consulted to compile a list of potential candidates for each type of security analysis (SAST, SCA, and DAST). The next step was to filter the options based on the criteria specified in **Table 2**.

Criterion	Description
Last version's release date	The tool's latest version release date should be less than six months old, ensuring active maintenance and up-to-date features and capabilities.
Last commit date	The tool's latest commit should be within the last 3 months, ensuring active development and timely addressing of bugs and vulnerabilities.
GitHub stars	The tool should have more than 1,000 GitHub stars, indicative of its popularity, large user base, reliability, and effectiveness.
Free access for private repositories	The tool should offer free access to its features for private repositories, thereby supporting secure development without incurring additional costs.
Comprehensive free features	The tool should incorporate as many free functionalities as possible, allowing the concentration of the needed capabilities within a minimalist toolset, thereby promoting efficiency and cost-effectiveness.
Code privacy	The tool should not upload or share the source code externally, ensuring the privacy and confidentiality of the proprietary code.
CI/CD integration capability	The tool should be designed to seamlessly integrate into CI/CD pipelines, thereby promoting continuous and automated security checks during the software development life cycle.

Table 2 - Tool comparison criteria

Tool Comparison Results

Filling the security gaps in any organization requires careful selection and testing of tools that are compatible with the existing tech stack. In this context, multiple free tools were meticulously examined to address the specific security needs of Altice Labs.

This involved a comprehensive evaluation of tool features, compatibility, ease of integration, and their effectiveness in identifying and mitigating vulnerabilities. The goal was to assemble a toolset that could not only enhance Altice Labs' security posture but also integrate seamlessly with their existing development processes.

The outcome of this rigorous comparison and testing process is illustrated in **Figure 2**. This figure presents the security tools that were ultimately selected for inclusion in Altice Lab's security toolbox. The tools are organized according to the type of security analysis they perform, providing a clear representation of how they collectively contribute to a more secure and robust SDLC at Altice Labs.

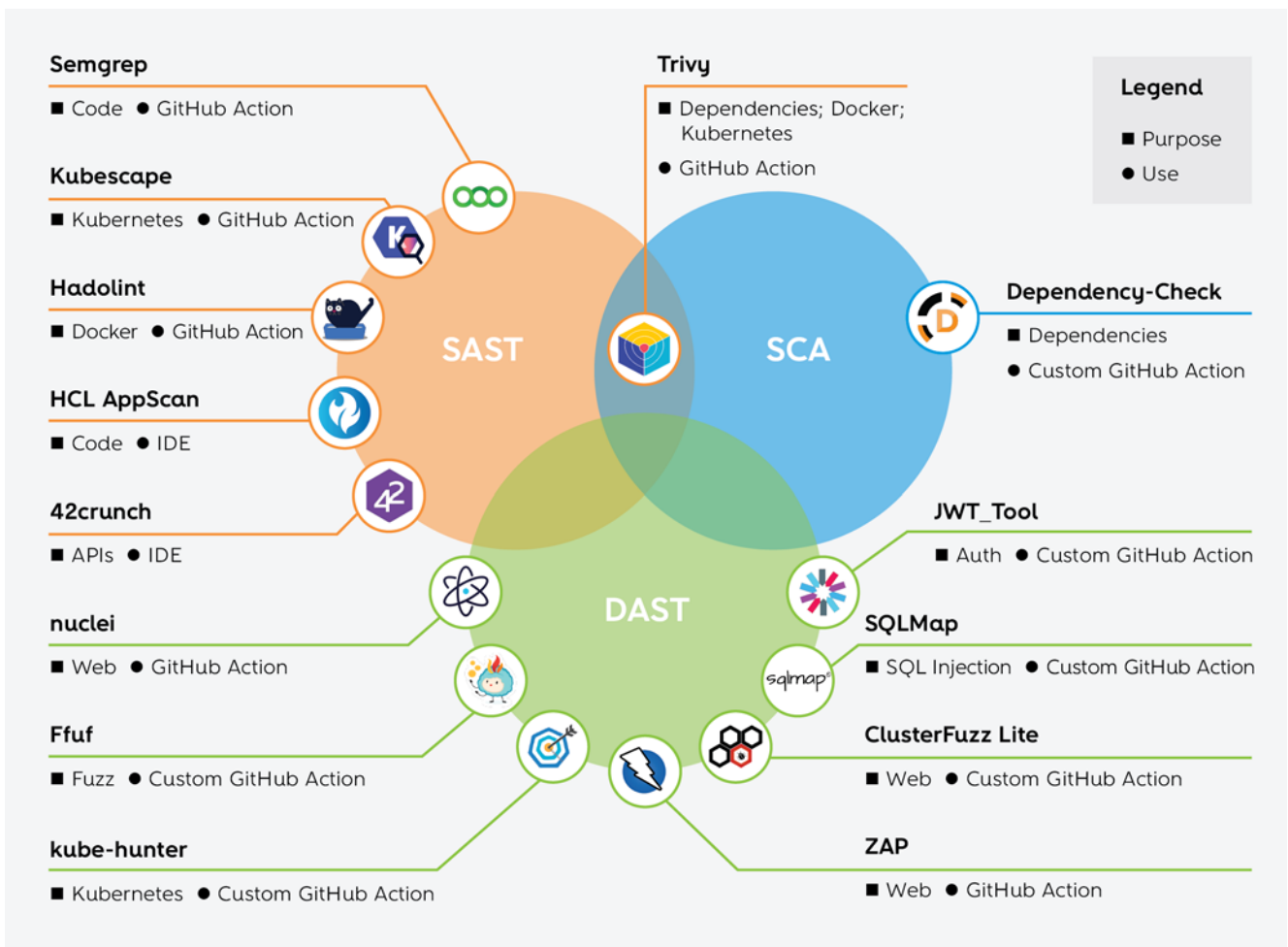
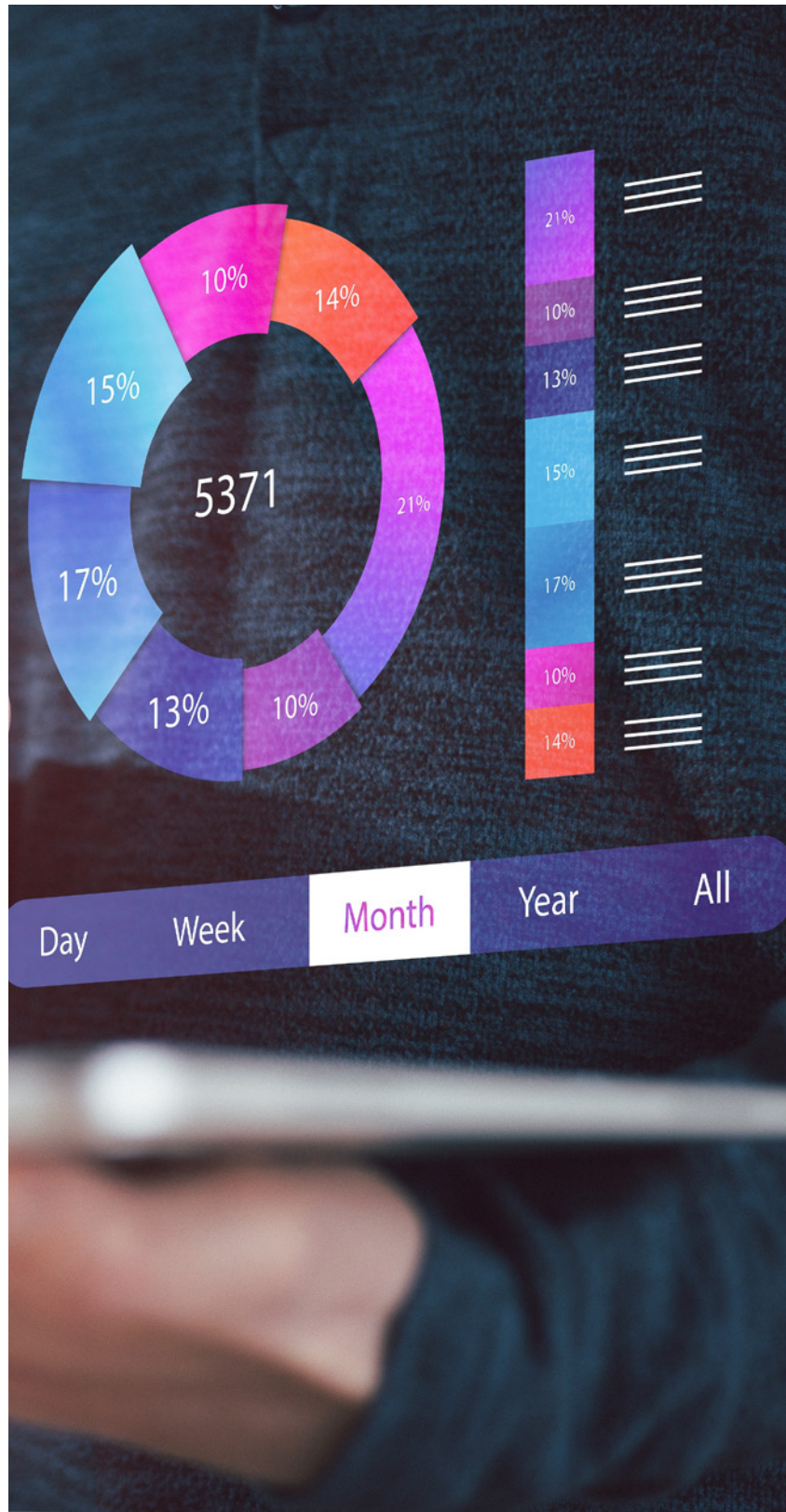


Figure 2 – Tool Comparison Results

SmartAL case study

In an enterprise context, creating a CI/CD pipeline is crucial for maintaining robust, reliable, and efficient development processes. GitHub Actions serve as an excellent resource to build these pipelines, enabling automation of software workflows directly in GitHub repositories [12]. Additionally, since SmartAL is one of Altice Labs’s projects that already use GitHub both as code repository and Docker images registry, GitHub Actions became an obvious choice. SmartAL is an Assisted Living product that offers a range of services, including teleconsultation for virtual doctor’s appointments and telelocation to monitor the whereabouts of elderly relatives. This complex application, with its intricate codebase and multiple dependencies, typifies the applications developed within the enterprise. The intention is to assess the pipeline’s effectiveness, detect any security vulnerabilities in SmartAL, and enhance the organization’s overall security posture. Importantly, the SmartAL version being evaluated (SmartAL-b2c) is under development, and detected vulnerabilities will be addressed based on their severity prior to production deployment.

When it comes to security, the CI/CD pipeline provides a critical framework for integrating automated security checks throughout the software development process. By embedding security checks within the CI/CD pipeline, enterprises can create a culture of continuous security that parallels and supports their continuous development and deployment practices. This integration enables early detection and remediation of security vulnerabilities, minimizes the cost and disruption of fixing issues late in the development process, and fosters the development of inherently secure software.



Aiming to increase the security of the software developed at Altice Labs, a pipeline was developed, incorporating a range of automated security checks, using the free security tools that were carefully selected earlier. SAST is performed in the first place, leveraging tools that systematically check for common vulnerabilities such as exposed secrets, SQL injections, or improper configurations that can lead to security breaches. The pipeline then moves to the SCA stage, using tools that help to highlight any known vulnerabilities in the dependencies and ensure compliance with open-source licenses, minimizing both security and legal risks. The last stage of the pipeline involves DAST. For this, tools that mimic the actions of potential attackers provide crucial insights into possible real-world exploits that the application might be susceptible to.

Following this detailed explanation of the pipeline, a graphical representation of the process has been provided for a clearer understanding. **Figure 3** concisely encapsulates this process, laying out the sequence of automated security checks from SAST to SCA to (deployment to) DAST. Each stage uses specific free security tools, systematically identifying potential vulnerabilities within the software, ultimately painting a comprehensive picture of its security profile.

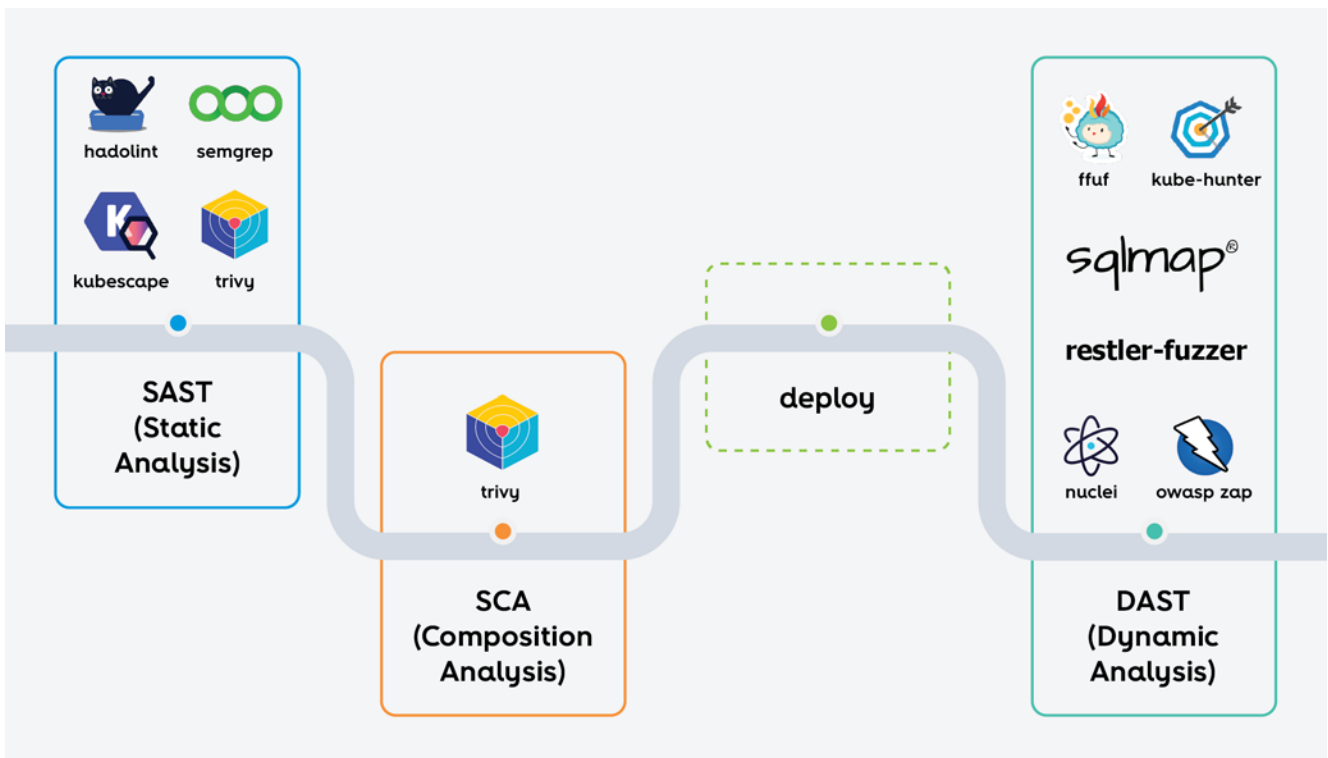


Figure 3 - Pipeline Phases and Tools

Implementation

In the implementation of SAST and SCA in the SmartAL-b2c repository, four primary tools were utilized: Hadolint for dockerfile linting, Semgrep as a customizable code-scanning tool, Trivy for comprehensive vulnerability scanning of containers and filesystems, and Kubescape for assessing the security posture of Kubernetes manifests. The automation was orchestrated through GitHub Actions workflows, where each tool composed a unique job encompassing repository checkout, tool execution, and report upload. Upon completion, a staggering 699 potential security issues were unearthed across both SmartAL-b2c and SmartAL-b2c-entities repositories. This extensive vulnerabilities list included dockerfiles bad practices, source code vulnerabilities, Kubernetes misconfigurations, and known vulnerabilities in open-source components and third-party libraries.

Following the rigorous SAST and SCA tests, the SmartAL application's deployment process was initiated within a Kubernetes environment. This encompassed steps such as repository checkout, setting up a single-node Kubernetes cluster through Minikube, generating a Docker registry secret using GitHub secrets to securely handle sensitive data like access credentials, and finally applying the necessary Kubernetes configuration files. The flawless execution of these steps attests to the merit of integrating automated deployment processes into the CI/CD pipeline, handling the intricacies of Kubernetes-based applications, and securely managing sensitive data. It served as a significant validation of the robust design and architecture of the SmartAL application.

Upon successful deployment, DAST was conducted, incorporating a broad suite of tools, such as OWASP ZAP and Nuclei for web-based application scanning, FFUF and Restler-fuzzer for fuzzing tests, Kube-Hunter for dynamic analysis of the Kubernetes cluster, and SQLMap for SQL Injection vulnerability detection. Configuring these DAST tools raised some challenges, especially due to the use of short-lived JWT tokens by the SmartAL application, which was at odds with the usual extensive duration of DAST analysis. Despite the challenges, each tool furnished valuable insights. OWASP ZAP spotlighted 17 potential security issues, highlighting the significance of API (Application Programming Interface) security. Nuclei, while needing some refinement in JWT handling, was instrumental in identifying outdated WordPress plugins and a medium-severity CVE. FFUF, despite not unveiling significant issues, demonstrated its value in the process by discovering exposed URLs. When deployed inside the pod, Kube-Hunter identified multiple vulnerabilities, thus emphasizing the importance of varying the perspective of security scans. Notably, SQLMap found no SQL injection vulnerabilities, a testament to the effectiveness of the application's input sanitization and SQL injection prevention mechanisms.

Despite the hurdles faced during the configuration and implementation of DAST tools, their ability to detect potential runtime vulnerabilities underlined their vital role in ensuring applications' security. With enhancements like JWT token utilization or implementing custom interactions, these tools could offer a more comprehensive and accurate application analysis.

These findings highlight the indispensable need for a comprehensive security approach that judiciously combines both static and dynamic testing methods.


Conclusions

In the intricate landscape of software development and supply chain security, the adoption of a comprehensive DevSecOps approach emerges as a crucial strategy. This research study aims to provide a thorough guide for securing the SDLC and Software Supply Chain, as well as to foster a holistic understanding of DevSecOps, equipping organizations with the required knowledge to strengthen their defenses against potential security risks.

The selection of suitable tools is a cardinal part of this process, requiring rigorous standards and exhaustive analyses. The empirical validation provided by the Altice Lab's case study further underlines the importance of tool selection and highlights the efficacy of a well-executed DevSecOps approach. By integrating strategic tools into the development pipeline, Altice Labs was able to identify and mitigate a substantial number of potential security issues, thereby attesting to the robustness and effectiveness of DevSecOps.

However, the path to successful DevSecOps implementation is not solely a matter of effective tool selection; it requires a strong organizational commitment. To fully realize the benefits of DevSecOps, organizations must be prepared to invest time, resources, and energy into refining their development processes, fostering a security-centric culture, and ensuring continuous learning and improvement.

The evolving ecosystem of DevSecOps tools, combining open-source and commercial offerings, presents a myriad of options for organizations to tailor a security solution that best fits their needs. While open-source tools enable DevSecOps by providing accessibility and flexibility, commercial tools have a vital role to play in addressing specific gaps and offering a more comprehensive feature set. Thus, a balanced approach that thoughtfully integrates both types of tools can serve as an effective strategy to optimize DevSecOps implementation.

In conclusion, this research elucidates the multifaceted benefits of integrating DevSecOps into software development and supply chain processes. By highlighting the practical measures, potential challenges, and successful case studies, it serves as a beacon for organizations embarking on their DevSecOps journey. The future of secure software development hinges on the wider adoption of such robust practices, and we hope this work will contribute to accelerating that shift. 



References

-
- [1] R. W. Macarthy and J. M. Bass, "An Empirical Taxonomy of DevOps in Practice," in 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020, pp. 221-228, doi: 10.1109/SEAA51224.2020.00046
-
- [2] M. Jaatun, D. Cruzes, and J. Luna, "DevOps for Better Software Security in the Cloud Invited Paper," in Proceedings of the 2017 Conference, 2017, pp. 1-6, doi: 10.1145/3098954.3103172.
-
- [3] M. S. Khan, A. W. Khan, F. Khan, M. A. Khan, and T. K. Whangbo, "Critical Challenges to Adopt DevOps Culture in Software Organizations: A Systematic Review," IEEE Access, vol. 10, pp. 14339-14349, 2022, doi: 10.1109/ACCESS.2022.3145970.
-
- [4] Sonatype, "8th Annual State of the Software Supply Chain Report | Sonatype." [Online]. Available: <https://www.sonatype.com/state-of-the-software-supply-chain/introduction>. [Accessed: 27-Apr-2023].
-
- [5] IBM, "Cost of a data breach 2022." [Online]. Available: <https://www.ibm.com/reports/data-breach>. [Accessed: 10-Jan-2023].
-
- [6] CIS, "The SolarWinds Cyber-Attack: What You Need to Know." [Online]. Available: <https://www.cisecurity.org/solarwinds>. [Accessed: 18-Apr-2023]
-
- [7] M. A. Akbar, K. Smolander, S. Mahmood, and A. Alsanad, "Toward Successful DevSecOps in Software Development Organizations: A Decision-Making Framework," Inf. Softw. Technol., 2022, doi: 10.1016/j.infsof.2022.106894.
-
- [8] F. Mateo Tudela, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-A. Sicilia Montalvo, and M. I. Argyros, "On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications," Applied Sciences, 2020.
-
- [9] A. Fatima, S. Bibi, and R. Hanif, "Comparative study on static code analysis tools for C/C++," in 2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST), 2018, pp. 465-469.
-
- [10] N. Imtiaz, S. Thorn, and L. A. Williams, "A comparative study of vulnerability reporting by software composition analysis tools," in Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2021.
-
- [11] T. Rangnau, R. v. Buijtenen, F. Fransen, and F. Turkmen, "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines," in 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC), 2020, pp. 145-154.
-
- [12] "GitHub Actions Feature." [Online]. Available: <https://github.com/features/actions>. [Accessed: 06-Jun-2023].
-

Acronyms

2FA	Two-Factor Authentication
API	Application Programming Interface
CaC	Configuration as Code
CD	Continuous Deployment/Delivery
CI	Continuous Integration
CNAPP	Cloud-Native Application Protection Platform
DAST	Dynamic Application Security Testing
IaC	Infrastructure as Code
IAST	Interactive Application Security Testing
IDE	Integrated Development Environment
JWT	JSON Web Token
RASP	Runtime Application Self-Protection
SAST	Static Application Security Testing
SBOM	Software Bill of Materials
SCA	Software Component Analysis
SDLC	Software Development Life Cycle
SIEM	Security Incident and Event Management
SmartAL	Smart Assisted Living
SVN	Apache Subversion
URL	Uniform Resource Locator
WAF	Web Application Firewall
XDR	Extended Detection and Response

Authors

Pedro Souto

Master in Cybersecurity

Altice Labs, Portugal



pedromgsouto@gmail.com



<https://www.linkedin.com/in/pedro-souto-b1613b22b>

Mafalda Nunes

Engineer in security and privacy in the development of systems

Altice Labs, Portugal



mafalda-g-nunes@alticelabs.com



<https://orcid.org/0009-0009-6085-0385>

Paulo Bartolomeu

Assistant Professor at Department of Eletronics, Telecommunications and Computer Engineering (DETI)

University of Aveiro, Portugal



bartolomeu@ua.pt



<https://www.linkedin.com/in/paulobartolomeu>

Contacts

Address

Rua Eng. José Ferreira Pinto Basto
3810 - 106 Aveiro (PORTUGAL)

Phone

+351 234 403 200

Media

contact@alticelabs.com
www.alticelabs.com
