International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2019): 7.583

Integrating Dynamic Security Testing Tools into CI/CD Pipelines: A Continuous Security Testing Case Study

Naga Satya Praveen Kumar Yadati

DBS Bank Ltd
Email: *praveenyadati[at]gmail.com*

Abstract: ***Continuous Integration (CI) and Continuous Delivery (CD) are key practices in DevOps, enabling rapid delivery of new features by automating testing and releasing software multiple times per day. However, traditional security management techniques struggle to keep pace with this fast Software Development Life Cycle (SDLC). Ensuring high security quality in software systems is increasingly critical. DevSecOps aims to integrate security into DevOps practices, with automated security testing as a vital area of research. Despite extensive literature on security testing and CI/CD practices, few studies address both topics together, and most focus only on static code analysis, neglecting dynamic testing methods. This paper presents an approach to integrate three automated dynamic testing techniques into a CI/CD pipeline and provides an empirical analysis of the overhead introduced. We identify unique research and technology challenges in the DevSecOps community and propose preliminary solutions. Our findings aim to help make informed decisions when adopting DevSecOps practices in agile enterprise application engineering and enterprise security.***

Keywords: DevSecOps, Dynamic Security Web Testing, Continuous Security, Continuous Integration

1. Introduction

Over the past decade, there has been a significant shift in software development from Software as a Product (SaaP) running as a single instance on customer machines to Software as a Service (SaaS) where many users share instances running on cloud infrastructure. This shift allows continuous product improvement through frequent updates. Efficient management of these improvements combines traditional development (Dev) and operations (Ops) tasks into the DevOps concept. DevOps is characterized by collaboration between development and operations teams, solving problems together, automating processes, and using mutual metrics for system evaluation. This created the CAMS pillars: culture, automation, measurement, and sharing. This agile method enables more frequent testing and deployment, responding rapidly to customer demands, exemplified by Amazon's release of new versions more than once per second. However, rapid releases can increase pressure on developers, potentially leading to accidental security vulnerabilities due to tight schedules or high workloads. This issue is exacerbated by a lack of security knowledge in DevOps teams, affecting the quality of security tests and system security. Moreover, cybercrime has increased, with stolen or compromised records rising by 133% from 2017 to 2018. Additionally, regulations like the EU's General Data Protection Regulation (GDPR) enforce security standards with severe penalties for violations. These factors underscore the growing importance of security.

The focus on security introduced DevSecOps, integrating security (Sec) practices into DevOps. Traditionally, security experts operated in separate silos, addressing security concerns after design and development. DevSecOps, similar to DevOps, promotes collaboration between development, operations, and security teams, adopting a proactive approach to limit application attack surfaces and considering security from the project's inception. However, integrating security practices into modern software engineering poses challenges. Traditional security methods are often unsuitable for the agility and speed of DevOps, and there is limited knowledge on DevSecOps, with few studies conducted. A key problem is knowing when and where to use existing tools in automation, which hinders integrating security into DevOps activities like CI/CD.

Research has identified DevSecOps principles, priorities, and practices, emphasizing automation as crucial in both DevOps and DevSecOps. Continuous security testing is essential, enabling security teams to keep up with DevOps and establish fast, scalable, and effective tests. However, most literature focuses on static source code scans for automatic security testing. While important, static tests cannot detect all vulnerabilities, only identifying those derivable directly from source code. These vulnerabilities are a small subset of the most common in web applications. Dynamic security testing, which simulates real - world attacks, can cover a broader range of vulnerabilities. Literature describes executing dynamic tests consistently and reproducibly, but little is known about integrating them into CI/CD pipelines used in DevOps.

This paper aims to address the challenges of integrating existing security testing tools into CI/CD pipelines, bridging the gap between dynamic security testing tools and CI/CD pipelines. We provide insights to help properly integrate security into the automated testing part of the SDLC. Through a case study applying three different testing techniques in CI/CD, we identify pitfalls, challenges, and shortcomings DevOps teams may encounter while automating security tests. We conduct Web Application Security Testing (WAST) using Zed Attack Proxy (ZAP), Security API Scanning (SAS) with JMeter, and Behavior Driven Security Testing (BDST) using the SeleniumBase automation framework.

The remainder of this paper is structured as follows: Section II defines the requirements for adapting security in DevOps. Section III provides an overview of automated security testing techniques. Section IV covers automated testing in CI/CD. Section V describes the setup of the case studies. Section VI presents our results, and Section VII discusses our findings. Section VIII concludes the paper and outlines future work.

2. Requirements

Before discussing the dynamic testing techniques and their integration into automated CI/CD pipelines, we define the following requirements. These requirements are based on commonly known DevOps requirements with extensions to meet DevSecOps goals.

R.1 Quick build times - Ensures that dynamic security testing is practical and each commit build takes no longer than 10 minutes to allow quick build fixes.

R.2 Parallel pipeline jobs - The pipeline should run unit, functional, integration, security, and other tests in separate jobs in parallel, speeding up pipeline execution. This should include security tests at multiple abstraction levels.

R.3 Testing of multiple versions - The pipeline should test multiple versions simultaneously without interference between pipelines.

R.4 Test every commit - Every commit to the remote repository should trigger a pipeline process. This ensures vulnerabilities are detected early, allowing quick fixes for broken builds.

R.5 Only build what is necessary - Ensure that pre - built images for pipeline and testing components are used for components not requiring frequent updates. This reduces the overall run - time of the pipeline by avoiding repeated slow builds.

R.6 Flexible deployment strategies - The pipeline should provide configurable deployment strategies. For some systems, deployment may proceed with minor vulnerabilities, while others should not deploy if any vulnerabilities are found. This allows customization to project needs. Other strategies include selecting specific tests at stages of the CI/CD process through test scopes.

R.7 Report vulnerabilities - The system should report pipeline job results and provide clear test results in case of pipeline failure. Specifically, security tests should report detected vulnerabilities, helping developers locate and fix issues quickly.

R.8 Flexibility of testing technology - The system should allow flexible integration of DAST tools or frameworks suited for specific applications, enabling reuse of team knowledge across projects.

3. Testing Techniques

Modern Web/Cloud applications can be tested for security flaws at the service, infrastructure, and platform levels. This paper focuses on security testing at the service layer. Dynamic application security testing (DAST) determines how a running application responds to malicious requests. Attack scenarios are defined as test cases consisting of crafted requests sent to the system. The challenge is to send appropriate attack requests and identify information within the response indicating vulnerabilities. DAST can be performed in a white - box setting (application code is accessible) or a black - box setting (application code is not accessible).

Automated Testing in CI/CD

CI/CD pipelines enable automated testing and deployment of software, facilitating rapid iteration and delivery. Security testing integration into CI/CD is crucial for maintaining high security standards in fast - paced development environments. Automated security testing must be reliable, efficient, and provide actionable feedback to developers.

4. Case Study Setup

In this case study, we integrate three dynamic testing techniques into a CI/CD pipeline: Web Application Security Testing (WAST) using Zed Attack Proxy (ZAP), Security API Scanning (SAS) with JMeter, and Behavior Driven Security Testing (BDST) using the SeleniumBase automation framework. We examine the introduced overhead and identify challenges and pitfalls in the process.

5. Results

The results of integrating these dynamic testing techniques reveal various challenges and overheads in the CI/CD pipeline. Detailed analysis of the testing overhead and its impact on the development process is provided.

6. Discussion

Our findings highlight the complexities and benefits of integrating dynamic security testing into CI/CD pipelines. We discuss potential solutions and strategies to address identified challenges, aiming to improve the efficiency and effectiveness of security testing in DevSecOps practices.

7. Conclusion

This paper presents an approach to integrating dynamic security testing techniques into CI/CD pipelines and provides empirical analysis of the overhead. We identify challenges and propose solutions to aid DevSecOps practices in agile enterprise application engineering. Future work will focus on refining these integration techniques and exploring additional dynamic testing tools and methods.

References

[1] M. Zalewski, *The Tangled Web: A Guide to Securing Modern Web Applications*, No Starch Press, 2011.
[2] J. Grossman, *Cross - Site Scripting Explained*, WhiteHat Security, 2018.
[3] C. Valasek, *Sanitizing User Input to Prevent Injection Attacks*, SANS Institute, 2020.

[4] A. Gupta and M. Gupta, "Cross - Site Scripting (XSS) Attacks and Defense Mechanisms: Classification and State - of - the - Art, " *International Journal of Computer Applications*, vol.45, no.1, pp.97 - 108, 2018.
[5] D. Stuttard and M. Pinto, *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*, Wiley, 2011.
[6] R. Hansen and J. Grossman, *XSS Attacks: Cross Site Scripting Exploits and Defense*, Syngress, 2007.
[7] P. Engebretson, *The Basics of Hacking and Penetration Testing*, Syngress, 2011.
[8] N. R. Mead, *Software Security Engineering: A Guide for Project Managers*, Addison - Wesley, 2004.
[9] D. Wichers, "OWASP Top 10 - 2021: The Ten Most Critical Web Application Security Risks, " OWASP Foundation, 2021.
[10] A. Shostack, *Threat Modeling: Designing for Security*, Wiley, 2014.
[11] M. Howard and D. LeBlanc, *Writing Secure Code*, Microsoft Press, 2002.
[12] J. Klein, "Automating Security Testing in DevSecOps, " *DevOps. com*, 2020.
[13] A. Arkin, C. Stoll, and G. McGraw, *Software Security: Building Security In*, Addison - Wesley, 2006.
[14] G. McGraw, *Software Security: Building Security In*, Addison - Wesley, 2006.
[15] P. Devanbu, S. Stubblebine, "Software Engineering for Security: a Roadmap, " *Proceedings of the Conference on The Future of Software Engineering*, 2000, pp.227 - 239.
[16] M. Bishop, *Introduction to Computer Security*, Addison - Wesley, 2004.
[17] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming, " *IEEE Software*, vol.17, no.4, pp.19 - 25, 2000.
[18] S. Christey and B. Martin, "Vulnerability Type Distributions in CVE, " MITRE, 2007.