Hochschule
Augsburg University of
Applied Sciences

Tools for Security Testing in Continuous Integration Pipelines

Michael Wager
Faculty of Computer Science
University of Applied Sciences Augsburg
Augsburg, Germany
mail@mwager.de

***Abstract*—As the world becomes more and more connected and lots of devices running software to support all areas of human life, security increasingly gets more important than ever. Modern software development life cycles already make use of intelligent processes and tooling in order to release high quality products. This work researches current modern security tools with the goal of automation in order to help make software more secure while integrating into existing processes. The tools presented are structured into three categories of application security testing (AST): static, dynamic and interactive. It can be concluded that lots of tools and professional products exist in the market today and can be used quite easily.**

I. Introduction

Software development is still a very young craft. But we are writing the year 2022 and the amount of devices containing more and more complex software is growing rapidly. Comprehensive quality assurance, especially in the field of web application development, also could deserve a lot more attention. Nevertheless, while a few years ago it was quite common practice to edit files on a production server directly via FTP in order to deploy a fix or a change, today methods and processes to assure a certain level of quality are used quite often. One of that method is called Continuous Integration (CI). The idea is to provide immediate feedback to the developers directly after the integration of changes to a codebase. On every push to the repository a so called CI server starts executing a pipeline of predefined tasks to ensure high quality and to make sure the changes do not break existing functionality. Common tasks are static code analysis to ensure a clean coding style or to catch simple development mistakes or unit testing to make sure the codebase maintains a high quality standard.

As this is also still a very young process, sadly software security does not play a big role inside these processes. As the pandemic led to a way higher usage of software in order to stay connected with friends, family and coworkers and also recent attacks against critical infrastructure moving through the media, cyber security gets a greater attention in recent times.

In the context of the Security Development Lifecycle [1], one idea to ensure a great level of application security is to automate as much and as early as possible directly during development using automation techniques inside CI pipelines. This work tries to provide an overview of categories of continuous security testing together with a review of existing tools. The paper is structured as follows: First an overview of related work and some technical background is given before presenting a selected set of tools. After a discussion the paper will end with a conclusion.

II. Related work

This chapter presents related work researched in the context of this paper, analyzing their open questions or missing goals in order to get an overview of existing work in this area.

Security testing in continuous integration processes [2] is a Master's Thesis by Juha Kuusela and reviews existing security methods and tools in order to analyze their applicability into CI pipelines. It concludes that, among others, dependency verification makes great sense and can be automated pretty easy. The paper only covers a section of available technologies and therefore cannot conclude on a wide range of relevant languages.

Evaluation of the applicability of security testing techniques in continuous integration environments [3] is a Master's Thesis by Pontus Thulin. It explores how existing security techniques work in a CI environment and what level of security they can help assure. It focuses on agile methods in the field of web application development and more on techniques instead of tools. However, as it takes a more theoretical approach, it is missing a comparison of tools in

order to better conclude about its practicability.

Automated Security Testing Utilizing Continuous Integration and Continuous Delivery Technologies [4], a Bachelor's thesis by Pyry Koskela, presents an implementation of a containerized vulnerability scanning construct tailored to a specific client infrastructure. It presents an interesting final implementation but lacks a comprehensive comparison of more tools.

Web application security testing as part of continuous integration in .NET projects [5] is a Master's Thesis by Joona Immonen from 2015. It focuses only on security controls for continuous integration in the field of web application development using .NET projects and the Jenkins continuous integration environment platform. Due to focusing on only one programming language it lacks a greater comparison of tools for generic usage.

Adding security testing in DevOps software development with continuous integration and continuous delivery practices [6] is a Bachelor's thesis by Ella Viitasuo from 2020. It focuses on developing a CI/CD pipeline for modern software projects including automated security testing as a first step to integrate security into software projects. As it only focuses on the field of web application development and OWASP Top 10 it is obviously missing a more generic comparison.

Challenges and solutions when adopting DevSecOps: A systematic review [7] by Roshan N. Rajapakse et. al. is a study aiming on systemizing the knowledge about challenges faced by practitioners when adopting "DevSecOps" and the proposed solutions reported in the literature conducting a Systematic Literature Review (SLR). It provides a comprehensive overview of existing literature, challenges and solutions to problems in the field of automating security assurance and also identifies areas that need further research like how to automate traditionally manual security practices.

As the field of automating security assurance is pretty young, all of the reviewed papers also are quite up to date (oldest being from 2015). All of them follow a similar goal but mostly tailored for specific use cases of specific set of languages or technologies. The purpose of this work is to give an even more up to date overview of tools and comparison for a more generic approach in order to give teams a simple and straightforward guide on tools to use, challenges to face and finally suggestions which tools could be preferred.

III. Background

This chapter provides a quick overview of basic concepts the reader has to understand in order to further grasp the contents of this paper. It will focus on the concept of continuous integration which is an important part of the concept of "DevOps" (Development and operations) as well as current efforts to automate application security assurance in modern software development products ("DevSecOps").

A. DevOps & Continuous Integration

Back in the days of early software development it was quite common to develop a product for months or even years and finally start to integrate it - i.e. preparing the release of the product. Stakeholders and developers had no clue how long this could take and lots of problems where introduced with this kind of process. To solve this problem, the idea of integrating often was first introduced in the extreme programming software development process [8]. The idea is simple: after making any change to a software product, be it a new feature of a bug fix, implementing and testing it locally on a developer machine, it will be commited to the source code control repository. Once a so called CI server recognizes the changes, it starts building and testing it again but this time it may also contain changes from other developers. The CI server will execute a so called build, including a pipeline of tasks to be executed in order to check if no issues are found. If this build fails the developer making the change will get notified immediately. And this is the whole point: integration errors are detected immediately and can get fixed right away, making the integration process much more transparent.

Some practices necessary to make this work are:

- Usage of a single source code repository
- Automation of the build using a CI server - every commit should build the product on the integration server
- Adding tasks to the build pipeline like static code analysis and unit tests which are executed after building
- Notifying all relevant parties of the project about failing builds

Nowadays, CI is quite heavily used in modern development projects and there are also ideas and efforts taken to automate security assurance inside the build pipelines of a CI server.

B. DevSecOps

Traditionally, security assurance is a process done mostly at the end of the software development life cycle. Security testing methods like penetration testing will concentrate on finding vulnerabilities in already (or soon to be) released products. While this is good and needed, the concept of DevSecOps suggest that security assurance may be automated as well as things like unit testing inside the CI server build pipeline - all while the development process is still ongoing. This work will focus on presenting a selection of up to date tools to achieve that.

IV. Tools

This chapter gives a overview of current tools for continuous security testing. It is structured into three parts: static application security testing, dynamic application security testing and interactive application security testing.

A. Static application security testing (SAST)

SAST tools scan the source code, byte code or binary of an application without running it with the goal of identifying vulnerabilities before deployment. They can assure secure coding practices are followed and find potential vulnerabilities in the source code. It follows the idea of integrating security early into the software development life cycle (i.e. shift-left security). The following criteria should be taken into account when evaluating SAST tools:

- **CI integration**
  It is important that the tool supports automated integration into existing CI pipelines and provides support for generating reports in common formats like XML or JSON.
- **Scan speed**
  A major goal of continuous integration builds is speed. If a SAST tool takes very long this may kill productivity.
- **False positives**
  A common problem with SAST tools is the generation of many false positives, i.e. detecting flaws which are no flaws at all. Therefore a good SAST tool should have a low false positive rate.
- **Developer productivity**
  In order to make developers work with the tool it is necessary to generate reports with good explanations of the detected flaws. Developers often may not have sufficient security knowledge and therefore hints on how to fix the detected flaws are very much appreciated.

Additionally to these criteria it should be looked at language support, the model of delivery and pricing.

1) SonarQube [9]: SonarQube community edition provides vulnerability detection, code smell tracking, technical debt reviews and remediation, code quality history and metrics. It can be integrated with CI pipelines like Jenkins, Azure DevOps and more and it is possible to extend its functionality further using more than 60 community plugins. It can detect injection flaws and may also be integrated directly into IDEs.

Language support: Supports 29 languages
Delivery model: On-premises
Pricing: Community: free. Developers: from $150

2) WhiteSource [10]: WhiteSource SAST provides visibility to over 70 CWE types - including OWASP Top 10 and SANS 25 - in desktop, web and mobile applications. They are claiming to be ten times faster than other SAST tools and it integrates well with existing DevOps environment and common CI tools.

Language support: Supports 27 languages
Delivery model: Cloud
Pricing: Annual subscription based on the number of developers

3) Veracode [11]: Veracode analyzes source code and provides automated security feedback and is easy to integrate into all major CI tools and also IDEs. They claim to have good speed, a false-positive rate of less than 1.1 percent and provides tips for fixing vulnerabilities when they are found.

Language support: Supports 25 languages for desktop, web, and mobile applications
Delivery model: Cloud
Pricing: Not publicly available

4) Codacy [12]: Codacy provides insights about the code that go beyond security, including the current code quality of the project and its health over time. It integrates well with existing CI tools and claims to have a low false positive rate.

Language support: Supports over 40 languages
Delivery model: On-premises and cloud
Pricing: Open-source: free. Pro: $15 per user/mo

5) AppScan [13]: AppScan performs vulnerability checks and generates a report that includes remediation suggestions. It supports automation via API and good integration into common CI tools.

Language support: Supports 35 languages and frameworks
Delivery model: On-premises and cloud
Pricing: Not publicly available

All tools presented in this section have very good language support and integrate well with existing CI tools. It seems that they are very modern and fulfill the requirements for static automated security testing in a very good manner. Table I compares the missing criteria defined in the beginning of this section.

Table I
COMPARISON OF SAST TOOLS

	SonarQube	WhiteSource	VeraCode	Codacy	AppScan
CI integration	very good	very good	very good	very good	very good
Scan speed	fast	very fast	fast	fast	fast
False positive rate	low	low	very low	low	low
Developer productivity	developer training included	easy to use	provides tips for developers	good developer feedback	developer friendly

B. Dynamic application security testing (DAST)

DAST tools finds security problems in applications by seeing how the application responds to specially crafted requests that mimic attacks. They are also known as web scanners and the OWASP foundation refers to them as web application vulnerability scanners. This means, other than SAST, they need a deployed and running application or API to work. The following criteria should be taken into account when evaluating DAST tools:

- **CI integration**
  It is important that the tool supports automated integration into existing CI pipelines and provides support for generating reports in common formats like XML or JSON.
- **Vulnerability detection rate**
  This means how many relevant vulnerabilities (e.g. path traversal, SQL injection, XSS) are detected by the tool.
- **False positive rate**
  Same as with SAST, the generation of many false positives, i.e. detecting flaws which are no flaws at all is also a problem with DAST tools.
- **Security expertise (Developer productivity)**
  Results should be readable by people with less security knowledge (i.e. regular developers)
- **Customization support**
  DAST tools usually only catch simple vulnerabilities because they do not have any context or clue of the business rules of the application (black box testing). Therefore the possibility for customizing test cases to detect more complex vulnerabilities is a bonus.

1) NetSparker/Invicti [14]: Netsparker is a comprehensive automated web vulnerability scanning tool. Its claims to be the industry leader in the area of DAST and provides high detection rates and good integration into CI solutions. It provides good automation support, clear reports, has a high vulnerability detection rate and a very low false positive rate.

Delivery model: On-premises and cloud
Pricing: Not publicly available

2) Acunetix [15]: Acunetix is a tool for automation of vulnerability detection for websites, web applications, and APIs. It claims to be an intuitive and easy-to-use platform. High focus is placed on Single-Page Applications (SPAs), it provides integration into bug/issue tracking systems and has good CI integration support.

Delivery model: On-premises and cloud
Pricing: Not publicly available

3) AstraPentest [16]: AstraPentest is a combination of automated vulnerability scanning and manual penetration testing and has good detection rate of common vulnerabilities like SQL injection or cross site scripting. Integration of the scanner with CI should be pretty easy, false positive rate is at zero and customization is possible.

Delivery model: Cloud
Pricing: $99-$399 per month

4) PortSwigger BurpSuite [17]: BurpSuite is one of the most well known penetration testing tools and regularly used as a desktop application for manual testing. Nevertheless they are providing native plugins for Jenkins and TeamCity, as well as a generic driver for any other CI platform one wants to use. This way it is possible to add automated vulnerability scans into existing pipelines and configure rules for failing the build based on the scan results.

Delivery model: Cloud
Pricing: Community: Free, Professional: $399/user/month, Enterprise: $3999/year.

5) Detectify [18]: Detectify is a vulnerability scanner to scan web applications or databases. It uses real payloads simulating an attacker to detect common vulnerabilities like OWASP Top 10 and can be easily integrated into CI tools.

Delivery model: Cloud
Pricing: Starting at $50/month

6) OWASP ZAP [19]: OWASP ZAP is a well known open source security tool and like BurpSuite normally used as a desktop application for manual penetration testing of web applications. But in spite of that it supports automated integration through its API including basic or advanced scans and test results. Integration seems to be a little bit harder compared to other tools but it is open source and free.

Delivery model: On-premise
Pricing: Open-source and free of charge

7) Veracode [20]: Veracode also provides a DAST tool on order to automatically scan web apps or APIs from a single

interface. It claims to be very fast through parallel execution of target scanning. It claims to have good low false positive rate and good documentation of found vulnerabilities.

Delivery model: Cloud
Pricing: Not publicly available

Table II provides a comparison of the criteria defined in the beginning of this section.

C. Interactive application security testing (IAST)

Contrary to SAST and DAST tools, there is a a new technology called interactive application security testing, or runtime security testing, short IAST. IAST tries to combine the advantages of the other two methods while leaving out the disadvantages they bring like false positive rates or scan speed. IAST does not need a running application, it depends on automated functional test cases or manual testing to drive the application and an agent runs in the background trying to identify vulnerabilities in the execution path of the tests already run or while they are running. This means IAST is only applicable if there already exists a very high test coverage of an application. But this also means that every test run will be automatically converted to a security test also.

It is using so called agents within the application to test for vulnerabilities dynamically in as it runs. These agents are connected to the application directly and are analyzing while it is manually or automatically tested, identifying vulnerabilities in real time. They are using runtime security instrumentation to analyze applications (e.g. source code, HTTP traffic, backend connections) thus providing a comprehensive view of vulnerabilities. This technology allows for easier testing automation than the other application security testing (AST) methods presented above because for example SAST tools may take very long and DAST tools are often inaccurate.

1) Contrast Asses [21]: Contrast Security argues to be the leader in this area of security testing. It claims to revolutionize continuous vulnerability detection without false positives, to be easy to install, simple to use and highly scalable. It provides complete coverage of the OWASP Top 10 and beyond and is of course easily automatable into existing CI tools. As is runs from within the running application, it can also generate simple diagrams that illustrate the application's major architectural components (i.e. used frameworks or running components like a database), thus providing simple architectural overviews in order to identify flaws in the design of a software product.

Delivery model: Cloud
Pricing: Not publicly available

2) Seeker [22]: Seeker is an IAST tool claiming to focus on real vulnerabilities. They have a patented verification technology which validates found flaws in order to report if they are real vulnerabilities or not, thus almost eliminating false positives. It claims to be easy to install and integrate into existing CI tools, it provides detailed documentation of found vulnerabilities including sample code to fix the issues

Delivery model: Cloud or on premise
Pricing: Not publicly available

3) NetSparker/Invicti [14]: Netsparker, which was presented above in the section of DAST tools, also provides IAST functionality claiming to find more real vulnerabilities using their unique dynamic + interactive scanning approach.

4) Acunetix [15]: Also Acunetix, which was already presented, provides an additional tool (called AcuSensor) to combine with their DAST tool in order to provide more accurate detection results and better overall test coverage.

5) Veracode [20]: Finally also VeraCode provides IAST functionality making it a complete solution to cover all three AST methods using one platform.

V. Discussion

Three common application security testing methods are presented in this paper: static, dynamic and interactive or runtime application security testing. A comparison of both SAST and DAST tools is given and also the new and pretty promising method of interactive runtime security testing is presented together with some current tools providing this functionality. This chapter will discuss diversities of the tools and tries to give recommendations for teams which want to integrate these tools in order to benefit from the promising field of continuous security testing.

A. SAST

Related the comparison of SAST tools all the presented tools have very good CI integration. The winner related speed is clearly WhiteSource, whereas VeraCode has the lowest false positive rate. All tools seem to be developer friendly. As a suggestion it would make sense to go deeper into WhiteSource and VeraCode if a team wants to integrate SAST into their existing CI and DevOps workflow.

B. DAST

Related the comparison of DAST tools all the presented tools support CI integration, while especially Netsparker and Acunetix make it more simple and easy. Also, Netsparker has a very high vulnerability detection rate and together with AstraPentest a false positive rate of zero which makes these two tools very promising. Information related developer productivity was not always available but customization seems to be possible with most of the presented tools.

C. IAST

Research of IAST tools found two tools definitely worth looking into more deeply: Contrast Asses and Seeker. Both tools seem to offer a really comprehensive set of advantages related continuous security testing, especially because they are

Table II
COMPARISON OF DAST TOOLS

	NetSparker/Invicti	Acunetix	AstraPentest	PortSwigger BurpSuite	Detectify	OWASP ZAP	VeraCode
CI integration	very good	very good	good	good	good	ok	good
Vulnerability detection rate	very high	high	high	high	n/a	n/a	n/a
False positive rate	zero	low	zero	low	low	middle	low
Developer productivity	very good	good	good	n/a	n/a	ok	very good
Customization support	possible	n/a	possible/manually	possible	possible	possible	n/a

embracing the interactive real time approach called IAST. Not only they combine the advantages of SAST and DAST tools while removing their disadvantages, they also integrate well with the DevOps method and additionally bring other benefits like architectural overview or conversion of already written test cases to security test cases.

Also, an important method for scanning external libraries for security issues was not presented in this paper but for example Contrast Assess provide the functionality to automatically analyze external third party libraries in order to find issues in these used dependencies. This security feature was also declared a low hanging fruit in some of the related papers where also other tools are provided for this purpose.

Most of the tools presented in this paper focus on the field of web application development. For the area of industrial software components and embedded devices, more research could be conducted.

VI. CONCLUSION

The goal of this paper was to research current tools for security testing in continuous integration environments in order to automate security testing inside the software development life cycle. Therefore, a recherche of related work was conducted to first get an overview of work already done in this area. Although the related work is presenting a good amount of available tools, research has shown that the market has to offer more. Also, all of the reviewed papers talk about the missing security knowledge developers have in general and it was found that the reviewed tools of this paper tackle this issue with good documentation and even tips for fixing found issues like source code hints. Besides that, most of the tools presented here offer very good integration into the DevSecOps step making it easy to integrate into existing CI tools. It can be concluded that lots of tools and professional products exist in the market today and can be used quite easily, be it open source or paid services.

This work only provides a theoretical review and comparison, no practical case studies were conducted. Because of that, in order for teams to decide on which tools to integrate, more practical reviews should be done and is highly recommended. Each team is different and has unique requirements. All of the presented tools provide demos or very good documentation.

Related the three categories of application security testing, especially the interactive real time approach (IAST) seems to be extremely effective and accurate. Because of its real time analyzation of running applications it provides a more sophisticated approach compared to the other two methods. Being able to scan each line of code from inside and therefore having knowledge about the control flow and context of an application, much better results can be presented. This knowledge is pretty helpful. For example, using IAST it is possible to identify credit card numbers extracted from a database and alert when these credit card numbers end up exposed in a log file. Also the possibility of conducting an architectural analysis and therefore finding flaws in the design of software seems to be a very large advantage. It can be concluded that, also related to time and budget, it is definitely worth looking into the approach of IAST first.

Although automated tools surely do not take away the need for manual security testing like threat modeling, code reviews or manual penetration testing, they can still support teams on their way to create more secure software in a great way.

REFERENCES

[1] M. Howard and S. Lipner, "The security development lifecycle," 2006.
[2] J. Kuusela, "Security testing in continuous integration processes," 2017, https://aaltodoc.aalto.fi/bitstream/handle/123456789/27065/master_Kuusela_Juha_2017.pdf.
[3] P. Thulin, "Evaluation of the applicability of security testing techniques in continuous integration environments," 2015, https://www.diva-portal.org/smash/get/diva2:784545/FULLTEXT01.pdf.
[4] P. Koskela, "Automated security testing utilizing continuous integration and continuous delivery technologies," 2021, https://www.theseus.fi/bitstream/handle/10024/502952/Opinnaytetyto_Koskela_Pyry.pdf.
[5] J. Immonen, "Web application security testing as part of continuous integration in .net projects," 2015, https://www.theseus.fi/bitstream/handle/10024/103333/Immonen_Joona.pdf.
[6] E. Viitasuo, "Adding security testing in devops software development with continuous integration and continuous delivery practices," 2020, https://www.theseus.fi/bitstream/handle/10024/342349/Opinnaytetyo_Ella_Viitasuo_pdfa.pdf.
[7] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting devsecops: A systematic review," 2022, https://www.sciencedirect.com/science/article/pii/S0950584921001543.
[8] K. Beck, "Extreme programming explained: Embrace change," 1999.
[9] "Sonarcube," https://www.sonarqube.org/features/security/sast/.
[10] "Whitesource," https://www.whitesourcesoftware.com/sast/.
[11] "Veracode," https://www.veracode.com/products/binary-static-analysis-sast.
[12] "Codacity," https://codacy.com/product.
[13] "Appscan," https://www.hcltechsw.com/appscan/offerings/source.
[14] "Netsparker/invicti," https://www.invicti.com/plp/dast/.
[15] "Acunetix," https://www.acunetix.com/plp/dast/.
[16] "Astrapentest," https://www.getastra.com/pentesting/web-app.
[17] "Portswigger burpsuite," https://portswigger.net/burp/.
[18] "Detectify," https://detectify.com/.
[19] "Owasp zap," https://www.zaproxy.org/.
[20] "Veracode," https://www.veracode.com/products/dynamic-analysis-dast.
[21] "Contrast asses," https://www.contrastsecurity.com/contrast-assess.
[22] "Seeker," https://www.synopsys.com/software-integrity/security-testing/interactive-application-security-testing.html.