jamk.fi

Adding security testing in DevOps software development with continuous integration and continuous delivery practices

Ella Viitasuo

Bachelor's thesis
May 2020
Technology, communication and transport
Program in Information and Communication technology

jamk.fi

Description

Author(s)	Type of publication	Date
Viitasuo Ella	Bachelor's thesis	May 2020
		Language of publication: English
	Number of pages 48 + 2	Permission for web publication: X

Title of publication
Adding security testing in DevOps software development with continuous integration and continuous delivery practices

Degree programme
Information and communication technology

Supervisor(s)
Mieskolainen Matti, Saharinen Karo

Assigned by
Company X

Abstract

In Company X there was found a need for creating a starting point for security testing in software project. As modern software development is moving forwards DevOps and agile type of development this would need to be suitable for that. The aim was to develop a continuous integration and continuous delivery (CI/CD) pipeline to include security testing that could be adopted in DevOps software projects. The pipeline should advocate open source tools that could be accessible for most different size software development projects.

The idea of developing a CI/CD pipeline for modern software projects with integrated security testing came from acknowledging how security is often misunderstood as add-on feature in software instead of build in quality. Software is a huge part of modern society and many software stores and handles sensitive information. How to protect sensitive information, should be considered when designing the software.

Emphasizing the importance of starting security testing even with small software development projects and benefits it can offer, is one of the main elements of the thesis. This thesis discusses about understanding security awareness and small steps that could improve it in development process.

Adding small steps towards building more secure software does not require too much effort or money. Open source tools can offer starting point of understanding security better and developing more secure way of coding. Understanding most common vulnerabilities and how to identify them in the development phase should be standard and easily achievable mission in software development process.

Keywords/tags (subjects)
DevOps, DevSecOps, Security testing, Application security testing, OWASP Top 10, CI/CD

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Viitasuo Ella	Opinnäytetyö, AMK	Toukokuu 2020
		Julkaisun kieli Englanti
	Sivumäärä 48 + 2	Verkkojulkaisulupa myönnetty: X

Työn nimi
Adding security testing in DevOps software development with continuous integration and continuous delivery practices

Tutkinto-ohjelma
Information and communication technology

Työn ohjaaja(t)
Mieskolainen Matti, Saharinen Karo

Toimeksiantaja(t)
Yritys X

Tiivistelmä

Yrityksessä X huomattiin tarve kehittää ratkaisu erilaisille ohjelmistokehitysprojekteille tietoturvatestauksen malli. Kehitettävän ratkaisun tulisi tukea DevOps sekä ketterä kehitys malleja, jotka ovat vakiintumassa modernissa ohjelmistokehityksessä. Tavoitteena työssä oli kehittää jatkuvan integraation (eng. Continuous Integration) sekä jatkuvan julkaisu (eng. Continuous Delivery) (CI/CD) tietoturvatestausputki , jota pystyttäisiin hyödyntämään DevOps ohjelmistoprojekteissa.

Tietoturva on usein väärinymmärretty ylimääräiseksi, lisättäväksi ominaisuudeksi sen sijaan että se olisi rakennettuna ohjelmistoon, josta ajatus tietoturvatestausputken kehittämiselle lähti. Ohjelmistot ovat iso osa modernia yhteiskuntaa ja useasti ohjelmistot käsittelevät ja varastoivat henkilötietoja sekä erilaista arkaluontoisia tietoja. Miten arkaluontoista tietoa suojataan, tulisi käsitellä jo ohjelmiston suunnitteluvaiheessa.

Tietoturvatestaustaus voi tuoda pienissäkin ohjelmistoprojekteissa jopa suuria taloudellisia etuja. Opinnäytetyössä keskitytään tietoturvatestausputken kehittämiseen avoimen lähdekoodin työkaluilla, jotta tietoturvatestausta ei jätettäisi pois kustannussyistä. Tietoturvan ymmärrys sekä sen parantaminen pienillä askeleilla on yksi opinnäytetyön pääteemoista.

Pienet askeleet tietoturvallisemman ohjelmistokehitykseen ei vaadi liikaa ylimääräistä työtä ja rahaa. Avoimen lähdekoodin työkalut voivat tarjota aloituspisteen tietoturvatestauksen kehittämiseen. Yleisten tietoturvahaavoittuvuuksien ymmärtäminen ja niiden tunnistaminen ohjelmiston kehitysvaiheessa tulisi olla saavutettavissa oleva tavoite.

Avainsanat (asiasanat)
Tietoturvatestaus, DevOps, ketterä kehitys, CI/CD, jatkuva integraatio, jatkuva julkaisu

Muut tiedot (Salassa pidettävät liitteet)

Contents

Figures

Tables

1 Introduction

Software are a huge part of modern society and many software store and handle sensitive information, that could be forgotten. Web application development trend grows as for example online shopping is constantly a growing market. As web application development grows, web application vulnerabilities grow as well.

In 2011 Sony Pictures Europe systems where exploited by using commonly known SQL-injection attack. Injection is number one in OWASP Top 10 most common web application vulnerability. The breach cost Sony more than 600 000 USD. Sony has suffered from many injection attacks against multiple locations including Sony Pictures Russian, Sony Portugal and Sony Europe. (Arthur, C 2012; Greenberg, A, 2011)

The aim of this thesis is to create a pipeline that could be utilized in software projects as a starting point to improve security testing and developing more secure code. Developing software fast and more agile does not mean that there is no time for testing the security in the process.

As security testing is automated as part-of the software build process it becomes easier to adopt more secure way of developing, when the security status is checked after every change. This thesis presents open source tools that could be used while not creating extra costs for the project. Understanding basis of the vulnerabilities especially for those that web applications may be exposed to; it is important in order to learn how to avoid them. As many exploiting methods are more available for anyone to use, protecting the software from the most common vulnerabilities becomes more crucial.

Company X is a large international company that has multiple different size software development projects. In smaller projects there might not be room in the budget to use expensive commercial tools for testing. For Company X it was important to create security testing also with open source tools as they can be incorporated without additional costs to the project. The biggest problem for open source tools can be the lack of support or documentation. When learning to maintain these tools they can also create a great value in the future.

2 Research method and material

The aim of the thesis was to develop a security testing pipeline that would be suitable for modern agile and DevOps software development projects. For the purposes of saving costs while still improving security testing status, all tools selected should be open source. As the end result developed pipeline model could be utilized in different projects in the future.

One of the most important parts of the thesis was to gather information to develop a solution. Qualitative research method was used to gather the information. Qualitative research aims to develop a solution or create more understanding to the problem. Action research was not applied as the research method as the solution could not be tested in the production by the Company X before the thesis. (Kananen 2015, 29.)

Research material was gathered from electronic books and publication as well as certification material about software development models and software security testing. User documentations were used as a material for technical tools.

3 Software Development Models

Every software project, regardless of the size of it, has a development life cycle, referred as Software Development Life Cycle (SDLC). Software development process goes through the same steps but as the selected development model changes, the cycles they go through may vary. Software development process undergoes usually: (Dooley 2017, Chapter 2)

- Conception
- Requirements gathering/exploration/modeling
- Design
- Coding and debugging
- Testing
- Release
- Maintenance/software evolution
- Retirement

In different models, some steps might be combined and hard to tell apart. For example, in agile models when trying to develop frequent releases faster, many steps blend it together. (Dooley 2017, Chapter 2)

3.1 Waterfall

Waterfall software development model is based on completing every step of previous phase of development before moving on next one. In waterfall model testing starts after development is completed. Software development methodologies that practice sequential development model, like waterfall, typically require months or years to be deployed for customer. Waterfall has some downfalls, as you are supposed to finish previous development activity before moving on the next one. All the requirements should be set on before the development process starts. Creating all requirements means understanding everything that the customer wants. In practice this is challenging and usually there comes changes in the development process. (Olsen, Parveen, Black, Friedenberg, McKay, Posthuma, Schaefer, Smilgin, Smith, Toms, Ulrich, Walsh & Zakaria 2018, 28; Dooley 2017, Chapter 2)

3.2 Agile

Agile is more of a way of working and challenging the way of thinking in software development team rather than technical practices. The agile way of thinking might make the change in the process. Development and Operations (DevOps) is considered more directive than agile and agile way of thinking makes DevOps software development flow work. Agile varies from other software development approaches by its focus on people behind the work and how they work together. Teams should be self-organizing with the ability to figure out how to achieve given goals on their own. (Clokie 2017, 6.)

Agile comes with multiple different frameworks that helps teams to organize the work and are the most common things that comes in-mind when talking about agile: Scrum, Feature-Driven Development, Test-Driven Development, Sprints, Planning

Sessions, etc. These frameworks are just tools to implement agile software development in practice but are commonly mistaken from being agile itself. (Agile Alliance, Agile 101)

3.3 DevOps

Agile made development process faster but the delivery process could not keep up with this, which sparked the idea of DevOps. DevOps was the solution to bring development and operations together to achieve quicker development and delivery to production. The development side needs to understand the production system and its constraints and to do so it needs to communicate closely with the operations team. (Sharma 2017, 5-9.)

Adopting DevOps practices Sharma (2017, 11) states that there are few practices above others: continuous integration and continuous delivery." Without these two capabilities, there is no DevOps, and they should be considered essential to DevOps adoption, with all others being extensions, or supporting capabilities." (Sharma 2017, 11)

Adding security view into DevOps, DevSecOps

As in the past the security role of software development was placed in the final stage in the development. Nowadays when development processes are much faster and agile the security point of view needs to be added earlier to the development process. DevOps enabled fast and frequent development cycles but adding up to date security practices in place it can be more efficient. The term DevSecOps adds the security mindset from the beginning. Securing applications and infrastructure from the start. As well as in DevOps and continuous integration and continuous delivery practices, automation plays a role in this as well. Automating security tests enables agile and smooth workflow. Even with the plain term DevOps, the main goal has always been to keep security in mind in every step of the development lifecycle. When developing the software with security in mind from the start, it is important to evaluate risks. (What is DevSecOps? N.d.)

As todays software development focuses on creating greater scale and dynamic infrastructure, the key elements are containers and microservices. DevSecOps needs to focus on container security and securing microservices. What to add to CI/CD process from security point of view Red Hat (What is DevSecOps? Nd.) lists:

1. Integrate security scanners for containers.
2. Automate security testing in CI process.
3. Add automated tests for security capabilities into the acceptance test process.
4. Automate security updates, such as patches for known vulnerabilities.
5. Automate system and service configuration management capabilities.

4 Continuous integration and continuous delivery

As previously stated by Sharma (2017, 11) the core capabilities of DevOps are continuous integration and continuous delivery. In Figure 1 is demonstrated continuous integration, continuous testing and continuous delivery that leads to the final product. After software is released into the production continuous monitoring should be applied. (Sharma 2017, 11)

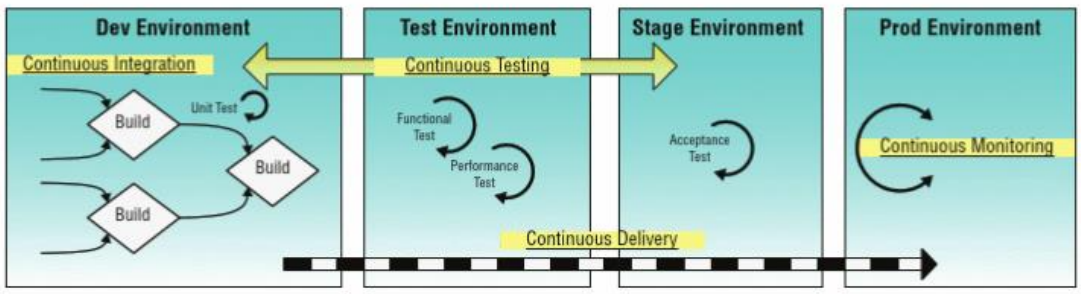

Figure 1 CI/CD process (Sharma 2017, 17)

4.1 Continuous integration

Continuous integration is considered to be one of the key technical practices from agile. In modern software development there are usually multiple people developing the code at the same time or developing different components of the software. To ensure that the separately developed parts work together they need to be integrated regularly, which is now known as continuous integration. The main goal of CI, besides

ensuring the developed software functions as expected, is to recognize dependencies between technology or scheduling. (Sharma 2017, 11-12)

For implementing continuous integration Sharma (2017, 13-15) includes a list of ten key practices listed by Martin Fowler (N.d.):

- Maintain single-source repository.
- Automate the build.
- Make your build self-testing.
- Ensure that everyone commits to the mainline every day.
- Ensure that every commit builds the mainline on an integration machine.
- Keep the build fast.
- Test in a clone of the production environment.
- Make it easy for anyone to get the latest executable.
- Make sure everyone can see what is happening.
- Automate deployment.

Maintaining single-source repository

Version management becomes essential when data needs to be accessed and modified by multiple different people in the project. Also, accessing data in multiple different locations and in different versions is often needed. Source code management, SCM, tool that supports multiple user access and versioning is critical in today's software development. (Sharma 2017, 13)

Automate the build

By automating build process, you ensure that the build contains everything it needs to succeed, and that the build is consistent (Sharma 2014, 32). Automating build process should also, if needed, be able to coordinate build to different platforms (Sharma 2017, 13).

Make your build self-testing

One of the goals of continuous integration is to ensure that the software that is built works as expected. Automating testing process, starting from the base unit-test level to the top application level tests ensures that the new changes did not break the

software. To test the software and deploy it after every build requires a lot of resources can also help to improve the quality of the code developed. Creating and maintaining automated tests, as well as to being able to run them at any given time, takes a lot of time and resources, that should be considered. (Sharma 2017, 14)

Ensure that everyone commits to the mainline every day

Committing changes, to earlier mentioned, common source code management system integrating work is easier to manage and not too complex. Integrating work regularly helps to identify risks and dependencies to other developers work. (Sharma 2017, 14)

Keep the build fast

One of the key elements in continuous integration and agile is the speed. Modern tools can help to increase the speed of the build process or build only new changes. (Sharma 2017, 14)

Test in a clone of the production environment

Testing in production like environment will give more realistic results for the end product and how it will function in production. For many reasons testing in production environment with production data, for example resources or need to mask the data, is not possible. Testing in a copy or similar environment will give some guidelines how the environment and its settings will affect on the software or application. In more complex systems or systems that are using pre-existing services, this might be challenging and creating test environment may create additional costs. (Sharma 2017, 14-15)

Make it easy for everyone to get the latest executable

Everyone in the development project should have access to the latest version that has been built and a way to utilize it. Having access to the latest built can be used to verify the changes made and are they working as anticipated. (Sharma 2017, 15)

Make sure everyone can see what is happening

Agile and DevOps both emphasize the collaboration between the people and the culture of sharing in the development project. The common encouragement of sharing

the information and improving visibility should come from the top of the organization. Sharing the visibility of the build status for all project team will give everyone involved a sense of the status of the software or application. In continuous integration sharing the build status is usually done by sharing the deployment pipeline or a dashboard. Extending visibility to all stakeholders, for example customers, creates an environment of sharing and working for a common goal. (Clokie 2017, 2-5, 44; Sharma 2014, 17; Sharma 2017, 15)

Automate deployment

Practising continuous integration often leads to adopting continuous delivery practises. Deployment tools are in the core of DevOps tools. Automated deployment tools make it easy to track the version of the build version that is deployed. Deployment automation tools can also usually manage environment configurations. Continuous delivery is more than just the deployment of the application. Continuous delivery practices more detailed in section 3.4. (Sharma 2014, 27)

For following these practices organization can find themselves practising more agile approach. Adopting continuous integration practises often leads to adopting continuous delivery practises. (Sharma 2017, 12)

4.2 Continuous testing

Continuous testing includes testing of the software or application being developed in every step of the SDLC. Continuous testing is often included on automated continuous delivery pipeline and continuous testing is often adopted, and included, in continuous integration and continuous delivery practices. Continuous testing is the response for more rapid software development where testing starts earlier in the software development, shift left approach, and in parallel with development activities. (Hollier & Wagner 2017, 3-6; Sharma 2017, 23)

A key aspect of continuous testing is to test early and often. Biggest challenges when it comes to continuous testing in agile or DevOps projects is the time. Testers might still be in previous iteration when development is focusing on the next one. Testing is often thought as too expensive as it often delays the delivery of the application when

time goes fixing the bugs testing finds. Also test environments and data might be issue in larger development projects. To handle these issues testing must evolve smarter. Most critical parts of the software should be tested as early and often as possible to ensure the business side. Test environment can include virtualization, or mock-services and they can be built up and tear down quickly. Including test automation into deployment pipeline can test the changes in the application and run tests more often. (Hollier & Wagner 2017, 21-23; Sharma 2017,25)

Shift left

Shift left approach focuses on moving testing earlier in the SDLC. The main goal of shifting left is to build in the quality and to find bugs earlier in the development cycle. Shifting left moves testing in the software build face and testing activities start before the whole application of software is deployed. This reduces the costs when bugs are found earlier when they are cheaper to fix than later in the production phase. (Hollier & Wagner 2017, 23)

Automated testing

When adopting DevOps and agile practices to improve speed of the development process it must be balanced with the quality of the development. With increased speed executing manual testing is not sustainable where automation comes in. Trying to automate everything is not possible or cost-efficient. Test automation process should start from evaluating which tests are efficient to automate. Developing sufficient test automation framework for software development process can be challenging if it is not approached correctly. Test automation is a software development project itself as it needs to have requirements, architecture, design, code and validation. Without careful planning test automation can become difficult to maintain and fragile which leads to its abandonment. (Hollier & Wagner 2017, 9-10, 16-17)

4.3 Continuous Delivery

Continuous delivery is automating software deployment process. Adopting continuous delivery process is one of the most important part of adopting DevOps practices. DevOps is larger scale than just continuous delivery, but continuous delivery is essen-

tial part of it. Deployment automation tools are the core tools of DevOps. Deployment pipeline that automates the process of build, test and deploy the application to environment makes continuous delivery actually continuous. (Sharma 2014, 21, 27; Sharma 2017, 16)

Deploying software manually is time consuming job. Most modern software are complex and manual deployment requires crafting the environment, installing required third-party components, copying data and configuration information. One of the biggest disadvantages in every manual installation is that depending on how the steps were followed it can lead to different outcomes which is rarely a good thing. Manually performed steps need to be documented in order to repeatable. (Farley & Huble 2010, 5-7)

Automatic deployment process is repeatable and is dependable only of the deployment scripts and not technical expert to handle every step of deployment process. Automated scripts acts also as documentation of deployment process. (Farley & Huble 2010, 5-7)

Deployment pipeline

Deployment pipeline is automated implementation of your projects build, deploy, test and release process. Every change that has been made, commits, needs to trigger the pipeline. Each test that runs in the pipeline are verifying that the new changes are working and can be released. (Farley & Huble 2010, 3-4)

There are three goals of deployment pipeline: (Farley & Huble 2010, 3-4)

1. Improves visibility of the build, test and deploy process for everyone involved
2. It creates a feedback loop to identify and solve problems quicker
3. It enables automated deployment and release of any version of the software to any environment

The deployment pipeline should include automated testing in different levels of the deployment process. Deployment pipeline should act as a tool to monitor the behavior of the software and the status of the deployment.

Clokie (2017, 44) included different level test types to be part-of the deployment pipeline:

- Static analysis of code quality
- Building the source code
- Unit testing, integration and test automation
- Functional and non-functional test automation
- Deployment scripts for different environments

Continuous Deployment

Continuous delivery could be described as a capability to deploy the software to the production or to any environment needed at given time, not actually deploying every change to production. Deployment in continuous delivery can also be parts of the application, components, and not necessary the whole software developed. Continuous deployment is actually deploying every change made to the production. Deploying every change does not mean everything that is deployed is a ready feature. Changes can be parts of the feature and may not be visible in the deployment at all. Continuous delivery also builds a version of software from every change, but the key is that it is not deployed to production. (Sharma 2017, 17-19)

One of the key things to remember of continuous deployment is that even if the change made could pass every test set in the deployment pipeline, it does not mean that the change is perfect. Automated test should be developed also during development process, as stated in section 3.2, and might not find the effects of the change. Acceptance tests should already be in place for the change before the change will go through the pipeline. When practicing continuous deployment test automation in all levels should be done with care and developed parallel or before the actual change is pushed to the pipeline. (Farley & Huble 2010, 266-267)

One of the upsides of continuous deployment is the risk related to releasing a software. Releasing software after every change, limits the changes in each release to just one. More changes in the release, more risks it contains. (Farley & Huble 2010, 266-267)

5 Testing security in software project

The main goal of software testing is to ensure that the developed software is functioning as it is supposed to. Software that doesn't work correctly may cause loss of money, time, business reputation or in works case injury or death. Testing process includes not only running tests but also planning, analyzing, designing, implementing, reporting progress and results and evaluating the quality of tested object. While testing can improve the quality of the software, testing doesn't mean that there are not any defects. It is not possible to test everything. Testing should be prioritized and planned, based on risk analysis and focusing on testing techniques, rather than focusing on testing everything. (Olsen et al. 2018, 13,16-17)

As with testing in general, the main goal of security testing is to minimize risks and improve quality of the security in the software. Security testing cannot prove that there will not be any vulnerabilities or that the application or software is safe from every attack there is. Security testing can be used to evaluate risks that the application or software have in securities point of view and evaluate the efficiency in security practices that are in place already. When it comes to software that handles sensitive data, there might be legal obligations when it comes to security practices. Neglecting security or protection of the digital assets in the software or application may result in legal actions. Security testing, however can help to prove that measures were taken to improve security practices and to protect those digital assets and may save from legal actions. (Rice, Daughtrey, Dijkman, Oliveira & Ribault 2016, 27-28)

5.1 Software security

In the modern world, software is essential part of critical systems. Most of security solutions are made to reduce the risks of insecure software. Whether to build security in the software or not is usually a business decision and evaluating costs of building the security versus the risks of not. One of the challenges on proving that building security is worth of the investment is explaining the technical vulnerabilities consequences to the business. Costs of insecure software is hard to estimate but Meucci and Muller (2014, 9) raised up a survey conducted by National Institute of Standards (NIST) that evaluates more than third of the costs could be saved if testing would

have been at better level. The goals of software security should align with the CIA principle:

- Confidentiality: Information is available to only those who are authorized to view it. Minimizing the unauthorized access is the goal of implementing confidentiality in the software.
- Integrity: Protecting the information's reliability and preventing unauthorized modifications.
- Availability: Authorized personnel and users have the ability to access the information at timely manner.

(Ransome & Misra 2018, 1-3; Chapple & Stewards & Gibson 2018; Meucci & Muller 2014, 9)

Ransome and Misra (2018, 2) also takes a stand when separating what is considered to be software security and application security, when they often are linked as the same: "In our model, software security is about building security into the software through a SDL in an SDLC, whereas application security is about protecting the software and the systems on which it runs after release." Whether the term for some may mean the same or vary as Ransome and Misra propose, based on Gary McGraw's description of both, most agree on security needs to be considered while developing software and different security activities must align in software development lifecycle.

5.2 Mindset of security testing

Security testing as well as testing in general should be part of each step of the software development process. Finding bugs and vulnerabilities earlier in the SDLC the lower the costs of fixing it will be. Security training is essential for not only security testers, but for developers as well as new vulnerabilities will arise all the time. Security training will help to create a security mindset that allows to think as malicious user trying to attack the software. Thinking outside the box of normal processes is the key of understanding what an attacker might use to attack the software. (Meucci & Muller 2014, 11-12)

Automated security testing tools are made to perform routine tasks and find commonly known vulnerabilities. Automated security testing, as well as automated testing in general, can not think outside the box and does not replace manual testing. Automates tools can still be useful and results can be further analyzed by security testers. When selecting automated security testing tool, it is important to evaluate what is the tool made for and what is wanted from it. (Meucci & Muller 2014, 12)

5.3 Security testing techniques and testing methods

Testing can be static, which is testing without actually executing the code. Static testing can be manual examination for example code review, or it can be tool-driven automated static analysis or evaluation. Static testing has become an important part of security testing and it is usually incorporated continuous delivery pipeline. Static analysis and incorporating static security testing in the delivery pipeline ensures or should encourage developers to follow secure way of coding. Since static testing does not execute code, it can be started early in the software development. (Rice et al. 2016, 53,78; Meucci & Muller 2014, 19)

Dynamic testing requires execution of the code or other objective that is tested. In security testing context static and dynamic testing techniques may be challenging to categorize depending what is considered as a test object. Definition of static testing is testing when the system is not in operational mode. In case of dynamic security testing tools, they often also perform static scanning to some parts of the system under test but are considered dynamic testing when test object is considered to be the whole system. (Rice et al. 2016, 79)

Fuzzing is security testing method which inputs a massive amount of data to component or system. Fuzzing may detect buffer overflow and memory corruption. (Rice et al. 2016, 54)

Penetration testing is attacking the software as a malicious user would. Penetration testing can go beyond the software itself to actual operating system and network that is being used in the production environment. Penetration testing is in the final stages before the software is in production and vulnerabilities found in penetration testing can get costly. Automated penetration testing tools are made to automate

process. Penetration testing can also act as assurance that vulnerabilities found in earlier security test are fixed correctly. (Meucci & Muller 2014, 14)

Security testing activities can also be categorized depending on the approach of testing. When testing is based on the knowledge of the software and information is disclosed about the software for example structure or design, testing can be called white-box-testing. When information is disclosed about the software being tested, testing can target specific parts of the software and coverage of testing can be measured more carefully. Source code analysis is considered to be white-box testing since the source code is available for testing. Black-box testing is the opposite of white-box testing. In black-box, testing is not based on information about the software and it should not be disclosed. Black-box testing can be more time consuming and might not find the embedded vulnerabilities in the code level that could have been found in static analysis of the code. Penetration testing is usually considered to be black-box testing. Grey-box testing can be placed in between the white-box and black-box testing methods. In grey-box testing, the tester might have some knowledge about the software. (Rice et al. 2016, 53; Meucci & Muller 2014, 12,14, 19)

Security testing should utilize different testing techniques and methods to build sustainable security testing framework. Different methods and techniques used depend on the software development phase and they should always be synchronized. (Meucci & Muller 2014, 14)

5.4 Security testing in software development life cycle

Security testing is not dependent of the software development model. The security testing activities just happen in different cycles or stages when the development model changes. In agile or DevOps method security testing should circle around in the software development increments. The role of security testing is shifting from previously seen just as black-box penetration testing to actually testing security in each development phase. Black-box penetration testing is costly and can only be executed with ready software. Fixing vulnerabilities found on ready software can become expensive. (Meucci & Muller 2014, 24)

Security testing is seen now as actual process instead of the product it has been seen before. Built-in security is achieved only by security-oriented design and security testing throughout the software development process. Adding security point-of-view into each development phase, allows to comprehensive perspective of software security. Security testing should start as the whole software development process starts and end to the same as the development process. (Rice et al. 2016, 49; Meucci & Muller 2014, 10-12)

Microsoft security development lifecycle

Microsoft has developed a model to develop more secure software called The Security Development Lifecycle: "The Security Development Lifecycle (SDL) consists of a set of practices that support security assurance and compliance requirements. The SDL helps developers build more secure software by reducing the number and severity of vulnerabilities in software, while reducing development cost." Developed model starts from beginning at the SDLC and as it is promoting Microsoft's own tools to utilize in the process, the steps and practices can be used with different tools as well. Microsoft's SDL consist of 12 practices (What are the Microsoft SDL practices? N.d.):

1. Provide training:

   Everyone involved must understand the basics of security to know how to build it into the software. Training enforces security policies, practices, standards and requirements of software security. (What are the Microsoft SDL practices? N.d.)

2. Define Security Requirements:

   Security requirements should, if possible be planned during initial design and planning stages but continually updated depending on the functionality being developed. Security requirements might include also legal and industry related requirements, internal standards and practices. (What are the Microsoft SDL practices? N.d.)

3. Define Metrics and Compliance Reporting:

Defining the minimum acceptance criteria for security quality helps to hold teams accountable for it. Setting severity thresholds for security vulnerabilities, timeframe for fixing them and tracking for security related bugs helps reporting and measuring security related issues. Setting practices from the start of the project helps teams to understand the importance of risks associated to security. (What are the Microsoft SDL practices? N.d.)

4. Perform Threat Modeling:

   Determining the risks associated the software security helps to understand what kind of security features are necessary for the software. Threat modeling can be done at different levels of the software. (What are the Microsoft SDL practices? N.d.)

5. Establish Design Requirements:

   Defining the features with security aspect already thought out helps developers to implement features with security built in. Adding security to features in the end will usually be more complicated than consistently adding for example authentication and logging throughout the development process will provide more sustainable solution. It is important to understand used security solutions and what kind of protection they provide. (What are the Microsoft SDL practices? N.d.)

6. Define and Use Cryptography Standards:

   All data while in transit or stored should be protected from unauthorized disclosure. Decision of used cryptography method should be left to experts and to use encryption methods already used in the industry. Design should also allow to change the used method at any time if needed. (What are the Microsoft SDL practices? N.d.)

7. Mange the Security Risk of Using Third-Party Components:

   Many of the modern software is built by including third-party components. Evaluating the third-party-components used and the possible vulnerability they might have and what risks would that cause is important. Planning the

response when ever a vulnerability is found on third-party-component is critical. Additional validation of third-party-components might be worth a while depending on the impact it might cause if it is vulnerable. (What are the Microsoft SDL practices? N.d.)

8. Use Approved Tools:

   Project should publish approved tools and encourage to use the latest version of them and utilize the security functions in them if possible. (What are the Microsoft SDL practices? N.d.)

9. Perform Static Analysis Security Testing (SAST):

   Analyzing the code before compilation ensures secure code practices are being followed. SAST tools are usually integrated to the pipeline to identify vulnerabilities each time the software is built. Some of the tools can also be in developer's environment to help developer actively while coding. (What are the Microsoft SDL practices? N.d.)

10. Perform Dynamic Analysis Security Testing (DAST):

    Executing security tests on fully compiled and running software can give results that can't be found in static analysis. There are a lot of tools that can be integrated to the CI/CD process easily but there are also options that can be used to detect vulnerabilities, for example fuzzing. (What are the Microsoft SDL practices? N.d.)

11. Perform Penetration Testing:

    Penetration testing is used to discover potential vulnerabilities that are results from errors in code, configuration or possible from deployment. Penetration testing is performing similar actions as a hacker or other malicious user. Testing can consist of automated and manual code reviews to provide more holistic view of the security. (What are the Microsoft SDL practices? N.d.)

12. Establish a Standard Incident Response Process:

In modern world new threats arise constantly. By developing proper incident response process, reacting to new threats is clearer. The plan developed, should be tested as well. (What are the Microsoft SDL practices? N.d.)

Security testing activities before the development

Security testing should start from the begging of the SDLC. Actual software development process starts from the planning of the software. Planning often includes designing the software and gathering the requirements. Requirements might include legal regulations about the software. Requirements should always be documented so that they can be reviewed. Documentation gives guidelines and policies that can be followed and visited later when needed. Documentation can also be standards such as cryptography standard that is being used. Gathered documents should be reviewed to ensure that they are correct, complete and understandable. Threat modeling is important part of planning phase. Threat modeling is made based on design and architecture reviews and models. Discovering security flaws before development is most cost-efficient and changes to the software are easier to make. (Meucci & Muller 2014, 24; Rice et al. 2016, 49-50)

Meucci and Muller lists (2014, 25) security mechanism that should be checked for security requirement flaws:

- User Management
- Authentication
- Authorization
- Data Confidentiality
- Integrity
- Accountability
- Session Management
- Transport Security
- Tiered System Segregation
- Legislative and standards compliance (including Privacy, Government and Industry standards)

Similar practices can be found from both ISTQB Certification documentation (Rice et al. 2016) and OWASP Testing Guide 4.0 (Meucci & Muller 2014) with Microsoft SDL.

Microsoft SDL practices 1-7 all consist of practices that are before the actual development work starts.

Security testing in development

Security testing can start from the start of the implementation and development process. In security point of view, it is important to test how the designed requirements are met and how the actual security methods are implemented, for example authentication. Security testing should start from the lowest level of implementation before separate component are attached to each other. Code reviews are a common way of ensuring that the coding practices set are in use. Code review is usually manual inspection from another developer. Code review can be based on check-lists and many different vendors offer ready checklist to use, for example Microsoft Secure Coding checklist. (Meucci & Muller 2014, 25; Rice et al. 2016, 50)

Source code analysis is often used to validate the code quality and security features automatically. Also developing unit test and dynamic analysis to validate the functionality of actual security features often helps to identify possible vulnerabilities. When developing testcases for security functionality, it is important to add not only positive cases but also negative cases that should not work. Negative testcases can help to identify issues with error handling. Automated static analysis tools can be integrated with the development pipeline. Quality gate should be set to the tool used which will not build the software if it is not met. (Meucci & Muller 2014, 20)

After the software is build testing can test the full software and simulate attack scenarios. Dynamic testing can be done with manual testing of the software or with automates tools. Dynamic security testing requires knowledge about the software as well as the security. Dynamic security testing after component integration level can consist of possible attack scenarios that are implemented manually or with tools, more targeted attacks to certain vulnerabilities and testing with specialized techniques, for example fuzzing. (Meucci & Muller 2014, 21)

When mirroring back to Microsoft's SDL practices 8-10 are in development phase.

Security testing in sofware deployment

Before software is deployed to customer, penetration testing should be performed. Final configurations that are used in production should be tested for potential misconfigurations, for example minimum privileges, SSL certificates, and only essential services are used. Penetration testing activities can be performed in the user acceptance environment, but final penetration testing should be done in the production environment. (Meucci & Muller 2014, 25)

Microsoft SDL practice 11 aligns with OWASP testing Guide 4.0.

Security testing in software maintenance

While software is in production, vulnerabilities can still be found, and new attacks are developed. Software can also need fixes for the business logic and updates. As in development phase every change needs to be tested for security flaws and vulnerabilities, new changes need to be tested the same way. (Rice et al. 2016. 51)

Regression testing should ensure that the business logic and designed features work as they are supposed to. Security regression testing should ensure that the security status will match the security requirements. Regression testing for security status can be hard to verify since the vulnerabilities can be in any part of the production environment, everything from configurations, network, operating system to hardware. After maintenance work has any changes to any part of the production environment, for example the hardware, the whole environment should be tested for potential security flaws. Regression testing and health checks should be done in regular bases even if no new changes are made after last check. Establishing a change management process for testing the changes made in production environment is critical. (Rice et al. 2016. 56-57; Meucci & Muller 2014. 26)

6 Web Application Security testing and OWASP Top 10

As discussed in chapter 4.1, application security testing often refers to testing the security of the application that is developed and not the whole software including the environment, hardware, operating system, network and other aspects. Web applica-

tion security testing focuses the security of the application itself. The goal is to ensure that the application meets the defined security related requirements. Web application security testing cannot ensure that there are no weaknesses in the production environment which can lead to exploitation of the software. Web application security testing often relies on black-box testing methods. (Meucci & Muller 2014. 27)

In modern software development OWASP top 10 has become a standard for web application security. OWASP top 10 list 10 most common security vulnerabilities in web applications. OWASP top 10 was made to raise awareness of security vulnerabilities as the software industry grows. As modern software is becoming more independent from the operating systems and hardware, as new technologies rise with cloud computing, microservices and containers, new vulnerabilities rises. As OWASP top 10 supports most common web application security risks, following it to secure all top 10 vulnerabilities won't still ensure your application is completely safe, but it is a good starting point. (Ransome & Misra 2018; OWASP Top 10 -2017 N.d.; Hodson 2019, Chapter 8.)

OWASP Top 10 is an open source organization that creates application security tools and standards, books about security testing, secure code development and review. The goal of OWASP Top 10 was initially to raise awareness to developers about the most common vulnerabilities but through time it has become a standard that more organizations use. 2017 Version of the OWASP Top 10 most common web application vulnerabilities (OWASP Top 10 -2017 N.d., 1-4):

1. Injection
2. Broken authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with known vulnerabilities
10. Insufficient Logging & Monitoring

Injection

Injection flaws occur when malicious data is sent to system as part of command or query, for example SQL, NOSQL, OS or LDAP injection. Interpreter can execute malicious command or access data without authorization. Injection can be easily discovered when analyzing the code. Malicious user can find potentially vulnerable system by using scanners and fuzzers against the system. Organizations can use static source code analysis and dynamic application testing tools to find potentially vulnerable parts of system. (OWASP Top 10 -2017 N.d. 6-7)

SQL-injection is most commonly known injection attack as SQL is considered to have easy syntax which explains why it is so commonly used. As SQL injection is most commonly known injection attack, there are more vulnerabilities that injection attack can exploit. (Hodson 2019, Chapter 8)

Broken authentication

Broken authentication often occurs when authentication and session management is not implemented correctly, and malicious users are able to compromise passwords, keys or session tokens. System may be vulnerable to attacks against authentication if it for example permits weak passwords or system allows brute force attacks or other automated attacks. Using multi-factor authentication, creating password policies against weak passwords, limiting failed login attempts and using session management can prevent vulnerabilities in authentication. (OWASP Top 10 -2017 N.d., 8)

Broken authentication could also mean stealing credentials that allows malicious users to login as legitimate user. Credentials, cryptography keys, tokens and session identifiers can be stolen using man-in-the-middle attack or exploiting other vulnerability for example Cross-site-scripting. (Hodson 2019, Chapter 8)

Sensitive Data Exposure

Many application and service handle sensitive data, for example credit card information, health information or personal information. Passwords are also considered to be sensitive information. Legal regulations, for example GDPR, can also determine how data should be handled and protected, but every organization should determine

ways to protect data while in rest and when transferring. Sensitive data exposure can occur if application transfers data without encryption or if it uses weak cryptographic methods. Storing and handling sensitive data should be avoided if it is not necessary for business as it might be a target for malicious users. (OWASP Top 10 -2017 N.d., 9)

XML External Entities (XXE)

Applications or XML-based web services that evaluate external entities within XML documents can be vulnerable to attacks such as denial-if-service. Vulnerability is exploitable in applications that accepts XML directly, XML uploads or inserts data to XML documents which is parsed by XML processor from untrusted source. Using less complex data format and avoiding sensitive data serialization can help to prevent XXE vulnerability. SAST-tools can identify XXE, but also code reviews are useful to identify vulnerability in source code. (OWASP Top 10 -2017 N.d., 10)

Broken Access Control

Access control allows user to perform only needed actions in order to do their job. Failures in access control configurations can lead to unauthorized information disclosure and user performing actions that it should not be able to do, for example modifying or deleting data. Access control methods should be centralized in application. If access control uses JSON Web Tokens, (JWT), they should be invalidated after logout. Access control methods should be tested by developers and testers since access control is hard to test using automated tools. (OWASP Top 10 -2017 N.d., 11)

On businesses removing privileges and access from employees is commonly forgotten and can lead to employee having unnecessary privileges. Automating privilege removal process can help to follow minimum privilege strategy. (Hodson 2019, Chapter 8)

Security Misconfiguration

Security misconfiguration may occur in any level of the application: network services, platform, web server, application server, database, frameworks, code, virtual machines or containers or storage. Application could be vulnerable if security configura-

tion is not implemented in every level of the system, unnecessary features and settings are used or system contains default usernames and passwords for example. IoT devices are commonly known from insecure default configurations. Logging and monitoring solutions are important part of software security, but too informative and detailed error messages can give malicious user too much information about the application.

Automating software deployment using deployment scripts can help to prevent misconfigurations. For preventing security misconfiguration hardening should be done in every level of the application and minimal privileges and features used. Security misconfiguration can often detect security misconfiguration flaws. (OWASP Top 10 - 2017 N.d., 12; Hodson 2019, Chapter 8)

Cross-Site Scripting (XSS)

XSS failures occur when untrusted data is included in new web page without validation. In existing web pages user inputted updates via browser Application Programming Interface (API) that create HTML or JavaScript can also be the source of XSS failure. XSS vulnerability allows malicious user to execute scripts in other users' browser. The goal of XSS can be session hijacking or redirecting user to malicious site. (Hodson 2019, Chapter 8)

Cross-Site Scripting has three forms: Reflected XSS, Stored XSS and DOM XSS. Reflected XSS allows unvalidated and unescaped input to application or API as part of HTML output. Attack may allow malicious HTML or JavaScript to be executed in victim's browser. Usually attack requires victim to interact with for example a malicious link inserted to page. Stored XSS allows application or API to store malicious user input that is viewed later by another user. Stored XSS is considered to be high or critical risk. Application may be vulnerable to DOM XSS if applications JavaScript frameworks, single-page applications and API's handle attacker controllable data dynamically. To prevent vulnerability to XSS attacks development should use frameworks that escape XSS by design, for example Ruby on Rail and escaping untrusted HTTP request data. Automated tools can detect XSS vulnerabilities, but there are also tools for exploiting XSS vulnerabilities easily accessible. (OWASP Top 10 -2017 N.d., 13)

Insecure Deserialization

Insecure deserialization can occur when deserialization allows untrusted user input. Deserialization is converting data from storage or transit format, which is often low-level such as binary or in file format such as XML or JSON, to object format. Serialization can be used in applications for example HTTP cookies, API authentication tokens or databases. System may be vulnerable to insecure deserialization if it allows malicious or tampered objects to be deserialized. Insecure deserialization can lead to serious attacks such as remote code execution. Flaws in deserialization can be found using automated tools but validating it can require more human input. As many of the OWASP Top 10 vulnerabilities can be exploited using ready exploits and tools, deserialization can be more difficult to exploit, but if exploited, its consequences should not be understated. To prevent insecure deserialization is designing architecture to accept serialized objects from only trusted sources. (OWASP Top 10 -2017 N.d., 14; Hodson 2019, Chapter 8)

Using Components with Known Vulnerabilities

Software is built from components such as libraries, modules and frameworks. Components often run as the same privileges as the application itself, that can cause a problem if it contains vulnerability that can be exploited. Some vulnerabilities can be easy to exploit with ready tools, but some may require more work. Using a component with known vulnerabilities can also be a business decision. In that case it is important to have a plan for in case someone tries to exploit the vulnerability. (OWASP Top 10 -2017 N.d., 15; Hodson 2019, Chapter 8)

Monitoring, scanning and updating application in regular bases can help to identify where there are potential components with known vulnerabilities. OWASP has developed Dependency-Check that scans application for known vulnerabilities. Planning update cycles, removing unused components and using only components from official sources can help to prevent attacks impact on application. (OWASP Top 10 -2017 N.d., 15)

Insufficient Loggin & Monitoring

Logging and monitoring are an important part of incident detection and response. Logging and monitoring can be used to detect suspicious activity in application. Insufficient logging can also refer to unclear log messages of errors and warnings or storing them without backing them up. Logging and monitoring solutions should always be able to answer questions: who, what, when and where. Suspicious activities, for example failed login attempts, should always be logged and alerted to the system administrator. Developing auditing solution with integrity controls will help to prevent data tampering and deletion. An incident response plan with sufficient monitoring and logging functions, will reduce the time since suspicious activity is noticed and reported. In order to help identify if login and monitoring is enough, examining logs after penetration testing is required. (OWASP Top 10 -2017 N.d., 16; Hodson 2019, Chapter 8)

7 Developing Open Source Security Testing Pipeline

The aim was to develop security testing pipeline that could be integrated to deployment pipeline using open source tools to save additional costs. Also, it was important that the security testing pipeline is such that it could be able to detect potential OWASP Top 10 vulnerabilities. The pipeline itself was built to support modern software with container technology.

Open source security testing tools divide opinions as they have many advantages but also disadvantages. Open source tools are usually free and can save costs, but it may take a lot of time and technical skills to configure and maintain them. (Rice et al. 2016, 78-79)

7.1 Pipeline architecture

For the basis of the pipeline it was important to use platform that supported agile and CI/CD principles. The chosen pipeline platform is GoCD which is an open source project supported by ThoughtWorks Inc. When looking at the DevOps and agile prin-

ciples visibility in the development process is emphasized along with open communication. GoCD has clear pipeline structure that can be used to share visibility in the developing process. GoCD pipeline view can be seen in Figure 3.

The pipeline was created for demonstrating purposes using docker environment. Used docker containers for testing can be seen in Appendix 2. The pipeline included static analysis with SonarQube, Clair-scanner and Dependency-Check and dynamic analysis and penetration testing with ZAP. The pipeline architecture pictured in Figure 2.

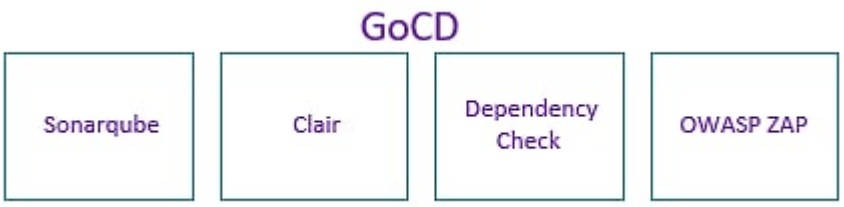

Figure 2 Pipeline architecture

GoCD

Figure 3 GoCD user interface

GoCD is specialized for CD but it can be used to CI as well. There are plenty of documentation and installation guides available in GoCD home page (GoCD User Documentation). There are multiple options when choosing a CI/CD platform and often tools support each other. As many most popular and used CI/CD tools list Jenkins very high it has some downfalls why it was not chosen for this. Jenkins is mainly designed for CI and tough adding CD is some cases possible it might be difficult to implement. GoCD pipeline structure was clearer and the view of the pipeline is clearer compared to Jenkins pipeline view in Figure 6.

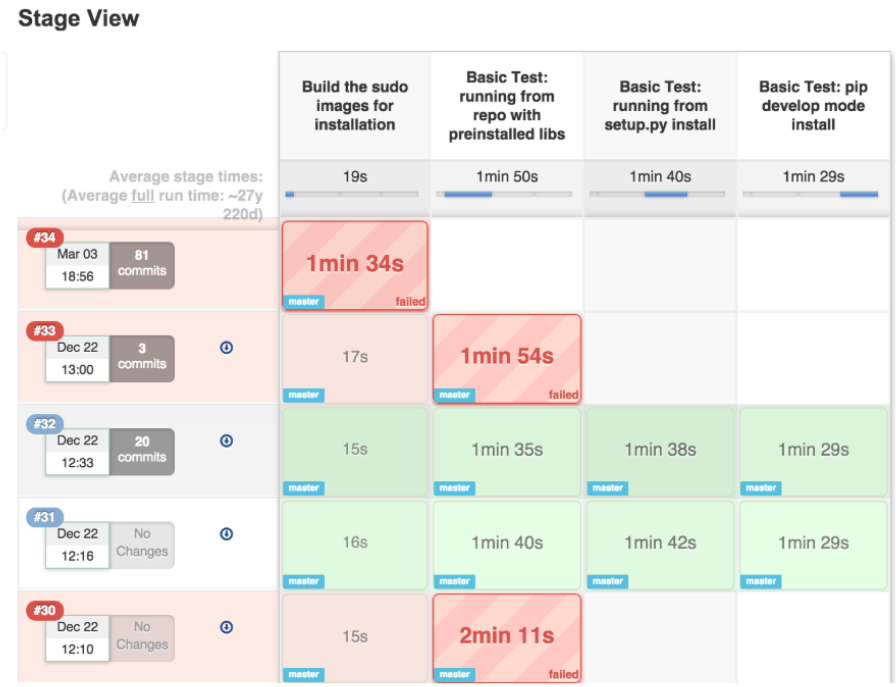

Figure 4 Jenkins pipeline view (Jenkins Official Page)

GoCD is also easy to integrate with other tools as it is as Jenkins mainly provides plugin to integrate other tools. GoCD has a user interface that makes configuring it easy, but pipelines can also be configured using YAML configuration files. (The differences between GoCD and Jenkins)

GoCD single pipeline view can be seen on Figure 5. It includes checkmarks for each passed step. The pipeline it self consists of 5 steps:

- Quality Check and code analysis
- Build of the application
- Vulnerability scanning of the container with Clair
- Checking vulnerabilities from 3rd party dependencies
- Vulnerability analysis and penetration testing.

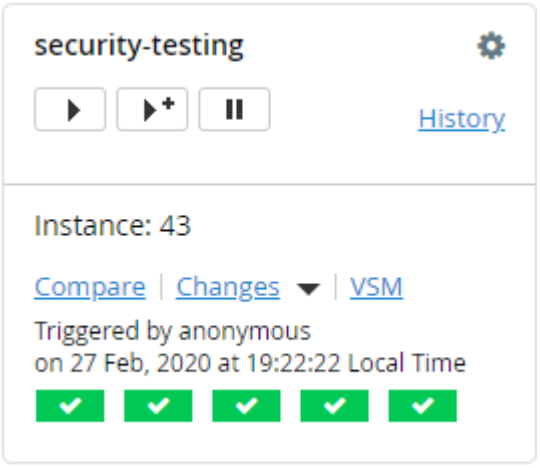

Figure 5 GoCD pipeline view

GoCD Agents

One of the things that makes GoCD easy to integrate is that you can build your own agents to do exactly what you need to. Agents are the ones that perform the actions in the pipeline. Multiple pipelines can be run simultaneously using multiple agents. Agents can be build based on needed quality and can have pre-installed content for the jobs. For this specific pipeline the agent needed to have Docker, Maven, Sonar-scanner, Clair-scanner and Dependency-Check-client and Java installed to run the jobs.

GoCD agent was built from Dockerfile based on Ubuntu 16.04 GoCD Agent Image. Installing needed tools for Dockerfile can be seen in Appendix 2.

7.2 Static analysis architecture

As in chapter 4.3, Security testing techniques and testing methods stated that static security testing has become a big part of security testing to ensure that the code is following secure code practices. Static security testing can be done often and early on software development life cycle. Static testing can start in code analysis level when the first lines of code are written and move on the container scanning when containers are built.

When selecting SAST (Static Application Security Testing) tools it was important that the tool could be automated and could detect OWASP Top 10 vulnerabilities. Also, important selecting criteria was that the tool would be open source and no extra

costs would be created to the project. Static analysis included also container security scanning to support security in container-based software.

SonarQube

SonarQube has taken steps to become more SAST tool as known for being a code quality scanner. SonarQube has listed in its documentation that it has security rules that supports CWE, SANS Top 25 as well as OWASP Top 10. SonarQube also has a plugin available to include OWASP Dependency-Check into SonarQube scan. SonarQube requires a language plugin to detect issues from the code but it supports a wide variety of coding languages. SonarQube has also commercial version that has more features. (Security-related Rules; Code Security, for Everyone)

To scan the code, you need both SonarQube server and the scanner running on the agent to perform the actual analysis. SonarQube server was installed as docker image that can be seen in Appendix 2.

SonarQube scan was run by installing sonar-scanner to a custom GoCD-agent build to run the pipeline. Sonarscanner installation to Dockerfile can be seen in Appendix 1. After the custom agent was built once, it could be used to run scans. After sonar-scanner was installed to the agent it could be called by using bash command: *>/sonar-scanner/bin/sonar-scanner -Dsonar.projectKey=key -Dsonar.sources=. \-Dsonar.host.url=url \-Dsonar.login=token*. Installing and configuring SonarQube was relatively easy and it could be controlled by using the SonarQube server UI. SonarQube UI report view can be seen in Figure 6.

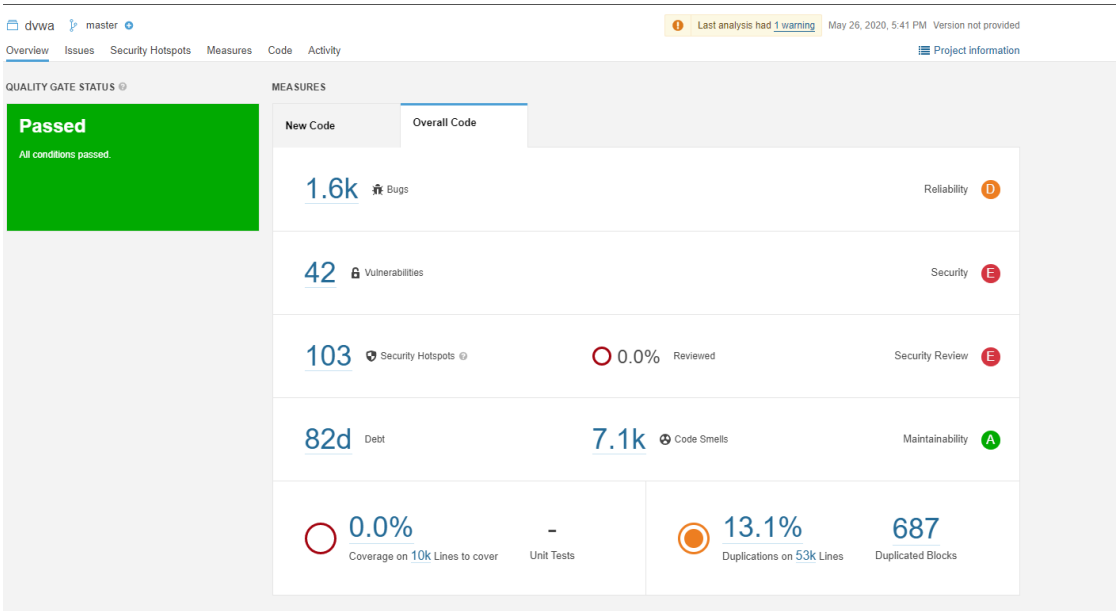

Figure 6 SonarQube report in server

SonarQube was chosen tool as it is easy integrate in various tools, for example GitLab, and it detects OWASP Top 10 vulnerabilities as well as analyzes code quality.

SonarQube endorsers secure way of coding by requesting user to review potential security vulnerabilities. Security review request, in Figure 7, gives option to mark as done or suppress the issue if the vulnerability could not be detected or exploited after testing.

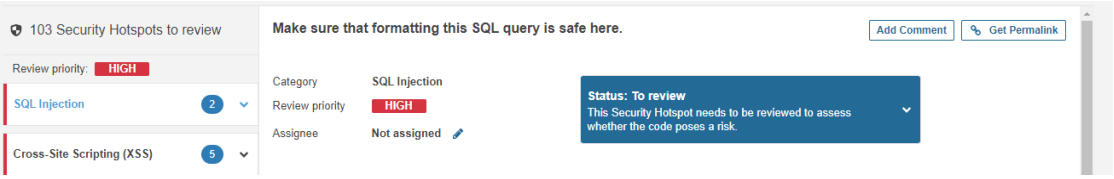

Figure 7 SonarQube warning from security vulnerability

CoreOs Clair

Clair is developed by Core OS and is open source container vulnerability scanner. It supports static vulnerability analysis for appc and docker containers. It can be integrated to pipeline build process of the application. For the desired quality, there is option to set the severity level of the vulnerabilities found that fails the scan if the desired quality is not met. This feature can be used to fail the application build if it contains critical or high-level vulnerabilities. Clair scans container layer by layer and searches for known vulnerabilities. (Clair 2.0.1 Documentation)

In this integration clair-scanner was used since it allows white-listing for approved vulnerabilities. Clair can be integrated different ways to fit the desired purpose. As in SonarQube you need to have both Clair server running, and the Clair-scanner installed to agent to be able to run scans against the container. Clair also needs a database that contains vulnerability information. Clair server image used can be seen in Appendix 2. Clair-scanner installation to the GoCD Agent can be found in Appendix 1. Clair-scanner was run from command line: *--clair=http://clair-server:6060 --ip=$HOSTNAME --threshold='Critical' --report=clair_report.json --log=clair_log.txt $image_to_be_scanned* .(Clair 2.0.1 Documentation; Clair Integrations; Clair scanner)

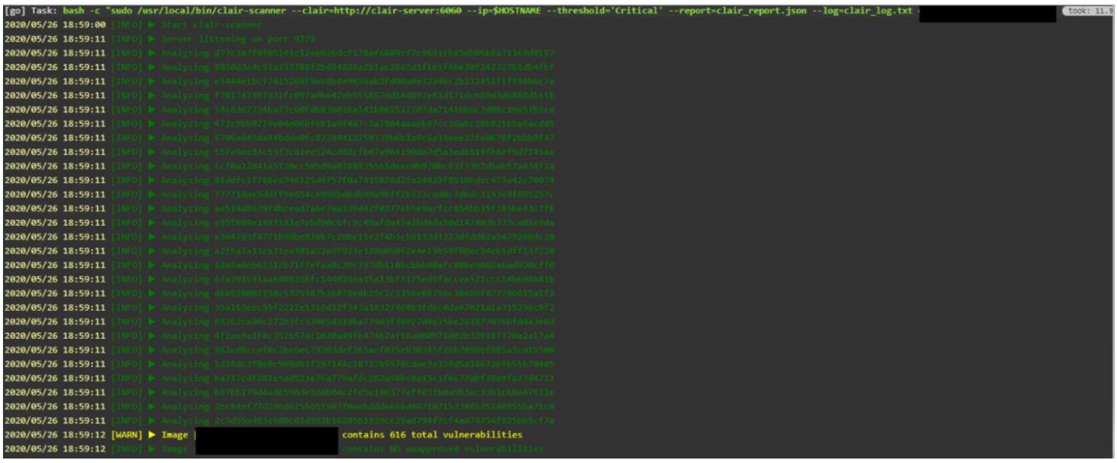

Figure 8 Scanning container with Clair

Clair is also used in GitLab own CI/CD tooling as an option (Container Scanning).

OWASP Dependency-Check

OWASP Dependency-Check identifies and detects dependencies that have publicly known vulnerabilities. Tool was chosen to identify components with known vulnerabilities that is one of the OWASP Top 10 security risks for web applications. Dependency-Check checks Common Platform Enumeration (CPE) identifier for given dependency and will generate a report that links it to associated Common Vulnerabilities and Exposures (CVE) entry. (OWASP Dependency-Check)

Dependency-Check did not require a server running and installing as well as running the scan was quite easy. Dependency-Check also provides a report as seen in Figure 9, that can be printed out to be analyzed. Dependency-Check was installed to the custom GoCD Agent image that can be seen in Appendix 1. The scan itself was run by

using command line: *--project "$project_name" --scan "$project_directory" --out "$output_directory" --format "HTML"*

DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided cons
or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | Suppressing false positives | Getting Help: github issues

Project: DVWA

Scan Information (show all):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Wed, 27 May 2020 12:24:12 GMT
- *Dependencies Scanned*: 10 (10 unique)
- *Vulnerable Dependencies*: 2
- *Vulnerabilities Found*: 10
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
container-min.js		pkg:javascript/YUI@2.5.2	MEDIUM	5		3
yahoo-dom-event.js		pkg:javascript/YUI@2.5.2	MEDIUM	5		3

Dependencies

Figure 9 Dependency-Check report

7.3 Dynamic analysis architecture

As in SAST, in DAST it was important that selected tools could be integrated together and automated. When targeting tools to identify OWASP Top 10 vulnerabilities it was logical to favor tool that was developed by OWASP.

OWASP Zed Attack Proxy (ZAP)

As stated in chapter 6 Web Application Security testing and OWASP Top 10, most of the vulnerabilities are exploited trough the API. OWASP ZAP can be used for both DAST and for penetration testing as well. ZAP provides many functionalities that detect and identify OWASP Top 10 vulnerabilities. ZAP includes many features that can be automated and included in CI/CD pipeline but also supports manual exploration for security testers. For the purposes of this security testing pipeline ZAP was integrated to pipeline using its API.

OWASP ZAP Desktop User Guide lists all features on ZAP. For scanning the application ZAP offers for example:

- Active scan:

  Active scan realistically implements attack scenarios on a target application. Active scan should only be targeting an application which you own or have permission to target since it can exploit the application. There is an option to set rules regarding the scan, for example setting different thresholds for certain vulnerabilities. (OWASP ZAP Desktop User Guide N.d.)

- Passive scan:

  Passive scan scans HTTP messages but does not try to modify them. A passive scan should not slow down the use of the application and is safe to use. (OWASP ZAP Desktop User Guide N.d.)

- Spider:

  The spider discovers resources of the given site. The spider can identify hyperlinks from the messages and add them as a resource. (OWASP ZAP Desktop User Guide N.d.)

- Man-in-the-middle proxy

  If ZAP is used as a proxy it allows to see traffic and requests from web application and responses it receives (OWASP ZAP Desktop User Guide N.d.).

ZAP can produce HTML format reports that can be fetch using the API. The report format can be seen in Figure 10. In this case report was attached in the GoCD pipeline as an artifact for accessing report after every run easily. In the report ZAP lists found vulnerability, information about it and possible solutions to fix it, as seen in Figure 11.

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	5
Medium	3
Low	9
Informational	2

Alert Detail

Figure 10 ZAP report

High (Medium)	Remote OS Command Injection
Description	Attack technique used for unauthorized execution of operating system commands. This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and/or improper calling of external programs.
URL	[redacted]/vulnerabilities/exec/

Figure 11 Vulnerability detail

OWASP ZAP has a few settings to be mindful of when scanning the target application:

- If application has authentication it needs to be setup for the scan
- ZAP is more powerful and might get more results when using it as a proxy when performing regression tests
- ZAP has different modes: Safe mode, standard mode, protected mode and attack mode and the scanning results may vary depending which mode you are using
- When using attack mode, it is important to exclude targets or URL's you don't want ZAP to attack

OWASP ZAP Desktop User Guide offers a list how to find and detect OWASP Top 10 vulnerabilities using ZAP. Automated active scan can detect: Injection, Sensitive data exposure, XML external entities, broken access control, XSS, using components with known vulnerabilities and insufficient logging and monitoring. For detecting Broken authentication and security misconfiguration manual inspection is required. Insecure deserialization detection is being developed to ZAP. (OWASP ZAP Desktop User Guide N.d.)

ZAP can be configured to run in the pipeline, but it may require a lot of work depending the type of your test environment and application. According to OWASP ZAP Desktop User Guide settings, ZAP available in public IP is only available when ZAP is running in AWS EC2 instance and setting public IP address to ZAP otherwise might

not work. ZAP does offer alternative solution to connect trough another proxy that might solve this problem for some. (OWASP ZAP Desktop User Guide N.d.)

ZAP image used for testing in this pipeline can be seen in Appendix 2. Zaproxy Documentation includes docker installation guide. Zap was executed in headless mode and could be started by using: *zap.sh -daemon -port $port_number -host 0.0.0.0 -config api.addrs.addr.name=.* -config api.key=$api_key -config api.addrs.addr.regex=true*

8 End Results

As an end result generated pipeline included status analysis, dynamic analysis and penetration testing steps as demonstrated in Figure 12.

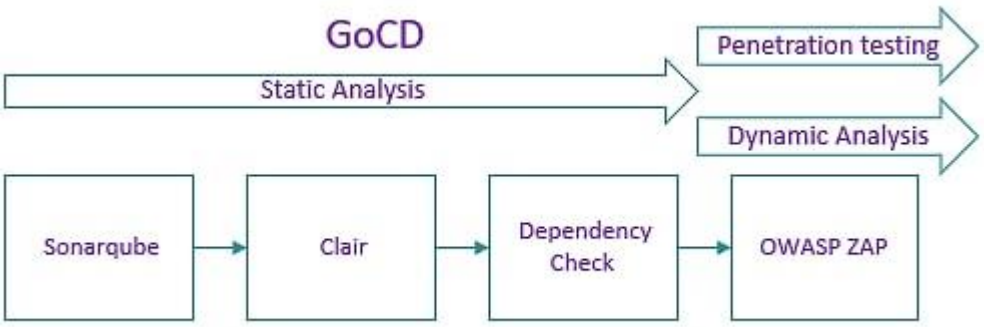

Figure 12 Generated pipeline

SonarQube found OWASP Top 10 vulnerabilities and other security and code quality related issues. SonarQube's updated functionalities support security issues better than the previous versions. SonarQube scan was tested against known vulnerable web application, Damn Vulnerable Web Application (DVWA). DVWA image that was used for testing can be seen in Appendix 2. DVWA was developed for testing web application vulnerabilities (Damn Vulnerable Web Application). SonarQube reports security related issues as security hotspots and vulnerabilities. SonarQube findings from the scan against DVWA can be seen in Table 1. Findings from security hotspots that could be categorized by possible OWASP Top 10 vulnerabilities can be seen in Table 2.

Table 1 SonarQube analysis reported vulnerabilities and security hotspots

SonarQube Report Security vulnerabilities and issues	
Vulnerabilities	42
Security hotspots	103

Table 2 OWASP Top 10 vulnerabilities found in security hotspots

SonarQube Found OWASP Top 10 in Security hotspots	
SQL injection	2
XSS	5
Insecure Configuration (Security Misconfiguration)	14
Weak Cryptography (Broken Authentication)	33
Remote Code Execution	3

Clair Scanner found total count of 616 vulnerabilities in the build docker container of DVWA. For testing purposes threshold for passing the scan was set to Critical vulnerabilities. High level vulnerabilities indicated that container could be vulnerable for stack-based buffer overflow, remote code execution, injection and man-in-the-middle attacks. Vulnerability count is demonstrated in Table 3.

Table 3 Clair-scanner vulnerabilities found

Clair Scanner Vulnerabilities found	Total Found: 616
High Level	12
Medium	315
Low	245
Negligible	44

Dependency-Check scanned for dependencies that could potentially have known vulnerabilities. Dependency-Check recognized two packages that could contain different known vulnerabilities. Report could summarize scanned dependencies and found issues that can be seen in Figure 13.

Summary

Display: Showing All Dependencies (click to show less)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
container-min.js		pkg:javascript/YUI@2.5.2	MEDIUM	5		3
high_unobfuscated.js				0		0
medium.js				0		0
high.js				0		0
yahoo-dom-event.js		pkg:javascript/YUI@2.5.2	MEDIUM	5		3
add_event_listeners.js				0		0
high.js				0		0
ConfigForm.js				0		0
impossible.js				0		0
dvwaPage.js				0		0

Figure 13 Dependency-Check summary

OWASP ZAP found 19 different vulnerabilities by scanning the application. Results from using the Spider and the Active scan can be seen in the report summary in Figure 14. Different types of vulnerabilities can be seen listed in Table 4. From OWASP Top 10 ZAP found traces of injection flaws, possibility for XSS exploitation, security misconfiguration and sensitive data exposure.

Summary of Alerts

Risk Level	Number of Alerts
High	5
Medium	3
Low	9
Informational	2

Figure 14 ZAP alert summary

Table 4 ZAP vulnerabilities found

ZAP Vulnerabilities found	Risk Level
Remote OS Command Injection	High
Remote File Inclusion	High
Path Traversal	High
SQL Injection	High
Cross Site Scripting (Reflected)	High
Application Error Disclosure	Medium
X-Frame-Options Header Not Set	Medium
Directory Browsing	Medium
X-Content-Type-Options Header Missing	Low/Medium
Cookie Without SameSite Attribute	Low/Medium
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low/Medium
Cookie No HttpOnly Flag	Low/Medium
Absence of Anti-CSRF Tokens	Low/Medium
Information Disclosure - Debug Error Messages	Low/Medium
Private IP Disclosure	Low/Medium
Application Error Disclosure	Low/Medium
Information Disclosure - Suspicious Comments	Informational/Medium
Timestamp Disclosure - Unix	Informational/Low

9 End discussion

The starting point for the thesis was to develop a pipeline using open source tools to suit different software development projects. Different aspects of developed solutions should be considered:

- How easy the tools where to install and use?

- Would the tools be suitable for company use?
- Did the tools provide results that could be acted on? Did the tools provide enough information about the status of the security?
- How the pipeline can be utilized in future?

Many of the tools used to build the security testing pipeline are easy to integrate and produce reports that can be utilized to further analyze the problems.

SonarQube was easy to integrate and maintain, which was expected as it is already popular code analyzer tool. SonarQube is actively developed and new security related features moves SonarQube closer to become a definite SAST tool. Clair-scanner was more work to integrate as there were not as much documentation available. Clair offers different integration options that could possibly provide better user experience. Dependency-Check was easy to use and integrating it did not require a lot of work. The difficulty of setting up ZAP depends on the testing environment and the desired effect it is used for. Setting ZAP as a proxy and using it during regression tests can prove to be quite powerful with low effort. The Active scans can be more stressful for the testing environment. ZAP API became helpful for automation, but it still is not the easiest tool to automate.

For open source tools lack of documentation and support can become an issue as commercial tools often offer more extensive documentation and user guide as well as customer support. SonarQube offers a lot of documentation available online in its official site. As SonarQube is a commonly used tool already, information is easy to find from other sources as well. Clair has commercial versions that could potentially have better documentation and a customer support. As for the clair-scanner integration version, it did not have much of a documentation available or that could be found. As Dependency-Check was easy to integrate and it offers different integration options, the lack of extensive documentation did not became an issue. ZAP required more studying to get better understanding of it, although a lot of useful information about managing ZAP can be found on OWASP ZAP Desktop User Guide.

Reporting functionalities on tools are important as customers usually demand the data from security testing also for themselves. SonarQube reports are clear and provide enough information about the issues found. Clair-scanner was not as corporate friendly as the JSON-format reports are not as appealing to look at or share to the

customer. The scanner found valuable information but there could be room for improving its suitability for larger scale projects. Dependency-Check HTML-format report was clear and suitable for potentially sharing it to the customer. ZAP provided HTML format reporting as well. ZAP reports contained information about the found vulnerability as well as for the possible solution on how to patch it. SonarQube, Dependency-Check and ZAP all provided information about the found vulnerability that could be utilized on evaluating the possible risks of leaving it or help to fix it. Security testing's goal is not to fix the vulnerabilities, but to give company or customer data that could help to evaluate possible risks that the software might be exposed to.

One of the most important goals set for the thesis was to see if these open source tools could find vulnerabilities and prove to be useful. As a source code material, this project used DVWA that was designed to be vulnerable, which explained why so many OWASP Top 10 listed vulnerabilities were found. As one of the main goals of security testing is to create understanding about the status of the software security, the selected tools provided a good view of the applications security status.

For the Company X developed pipeline could be utilized for smaller software development projects that does not have the budget for expensive commercial tools. As the researcher method for the thesis was qualitative, the developed solution does not contain case studies from production.

When looking at the developed model for testing security in software project, it is clear that only developing a pipeline that includes security testing at different levels cannot ensure that the software is secure. Developed solution consist of application security testing in development and testing environment. Penetration testing and application security testing should also be done in the production environment to find vulnerabilities in the environment itself that could expose the software at risk.

Establishing secure way of coding and understanding importance of security is the key of developing more secure software. Single pipeline cannot provide security training, but it can help to improve understanding inside the development project. OWASP Top 10 is not a standard of security and not finding any OWASP Top 10 vulnerabilities using developed pipeline should not be considered as an indicator of secure software. Eliminating OWASP Top 10 vulnerabilities from the software can be

seen as a starting point of making more secure software. The same method could be considered using this pipeline: it can be a starting point of security testing in the development process.

References

Agile Alliance, Agile 101. Accessed on 21.12.2019. https://www.agilealliance.org/agile101/

Arthur,C. 29.8.2012. LulzSec hacker arrested over Sony attack. Article on The Guardian. Accessed on 26.5.2020 https://www.theguardian.com/technology/2012/aug/29/lulzsec-hacker-arrest-sony-attack

Ashby, D. 2016. Continuous Testing in DevOps… Blog post. Accessed on 20.4.2020. https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/

Black, Claesson, Coleman, Cornanguer, Forgacs, Linetzki, Linz, van der Aalst, Walsh, & Weber. 2014. International Software Testing Qualifications Board. Foundation Level Extension Syllabus Agile Tester. Accessed on 21.12.2019

Chapple, M; Stewards J & Gibson, D. 2018. CISSP: Certified Information Systems Security Professional Study Guide, Eighth Edition Accessed on 12.4.2020

Clair 2.0.1 Documentation. Clair Documentation on CoreOs official site. Accessed on 26.4.2020 https://coreos.com/clair/docs/latest/

Clair Integrations. Github repository. Accessed on 28.4.2020 https://github.com/quay/clair/blob/master/Documentation/integrations.md

Clair scanner. GitHub repository hosted by Arminc. Accessed on 26.5.2020. https://github.com/arminc/clair-scanner

Clokie, K. 2017. A Practical Guide to Testing in DevOps. Accessed on 16.10.2019

Code Security, for Everyone. Sonarqube security features. Accessed on 26.4.2020. https://www.sonarqube.org/features/security/

Container Scanning. Gitlab Documentation. Accessed on 26.4.2020. https://docs.gitlab.com/ee/user/application_security/container_scanning/

Damn Vulnerable Web Application (DVWA). Information site about DVWA. Accessed on 27.5.2020. http://www.dvwa.co.uk/

Dooley, J. 2017. Software Development, Design and Coding: With Patterns, Debugging, Unit Testing, and Refactoring, Second Edition. Accessed on 30.4.2020.

GoCD User Documentation. Accessed on 26.4.2020. https://docs.gocd.org/

Hodson, C. 2019. Cyber Risk Management: Prioritize Threats, Identify Vulnerabilities and Apply Controls. Accessed on 23.4.2020

Hollier, M & Wagner, A 2017. Continuous Testing For Dummies, IBM Limited Edition. Accessed on 28.3.2020.

Humble, J & Farley, D 2010. Continuous delivery : reliable software releases through build, test, and deployment automation. Accessed on 30.3.2020

Greenberg, A. 20.6.2011. In Sony's 20th Breach In Two Months, Hackers Claim 177,000 Email Addresses Compromised. Article in Forbes. Accessed on 26.5.2020.

https://www.forbes.com/sites/andygreenberg/2011/06/20/in-sonys-20th-breach-in-two-months-hacker-claims-177000-sony-emails-compromised/#76f1caf94f06

Jenkins 2 Overview. Jenkins 2.0 Documentation. Accessed on 27.4.2020. https://www.jenkins.io/2.0/ Accessed on 26.4.2020.

Kananen, J. 2015. Opinnäytetyön kirjoittajan opas : Näin kirjoitan opinnäytetyön tai pro gradun alusta loppuun. Jyväskylä: Jyväskylän ammatikorkeakoulu. Accessed on 26.5.2020

Meucci & Muller 2014. OWASP Testing Guide 4.0 Accessed on 13.4.2020

Olsen, Parveen, Black, Friedenberg, McKay, Posthuma, Schaefer, Smilgin, Smith, Toms, Ulrich, Walsh & Zakaria 2018. International Software Testing Qualifications Board. Certified Tester Foundation Level Syllabus. 2018 Version. Accessed on 21.12.2019

OWASP Dependency-Check. Official OWASP page. Accessed on 27.4.2020. https://owasp.org/www-project-dependency-check/

OWASP Top 10 -2017.N.d. The Ten Most Critical Web Application Security Risks. Accessed on 23.4.2020. https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/

Ransome & Misra 2018. Core Software Security, Security at the source. Accessed on 12.4.2020

RedHat DevOps: What Is DevSecOps? Accessed on 30.3.2020 https://www.redhat.com/en/topics/devops/what-is-devsecops

Rice, Daughtrey, Dijkman, Oliveira, Ribault 2016. International Software Testing Qualifications Board. Certified Tester Advanced Level Syllabus Security Tester Version 2016. Accessed on 12.4.2020

Security-related Rules. SonarQube Documentation 8.2 Accessed on 26.4.2020. https://docs.sonarqube.org/latest/user-guide/security-rules/

Sharma, S. 2014. DevOps For Dummies, IBM Limited Edition. Accessed on 21.12.2019.

Sharma, S. 2017. The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise. Accessed on 16.10.2019

The differences between GoCD and Jenkins. GoCD Official Page. https://www.gocd.org/jenkins/ Accessed on 26.4.2020

Van der Stock, Glas, Smithline & Gigler 2017. OWASP Top 10 – 2017. The Ten Most Critical Web Application Security Risks https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

What are the Microsoft SDL practices? Microsoft official site. Accessed on 18.4.2020. https://www.microsoft.com/en-us/securityengineering/sdl/practices

OWASP ZAP Desktop User Guide. N.d. Accessed on 20.3.2020. https://www.zaproxy.org/docs/desktop/

Appendices

Appendix 1. Dockerfile configurations for building GoCD agent

Additional settings can be added to Dockerfile.

Base image for Dockerfile from Docker Hub:

- gocd/gocd-agent-ubuntu-16.04:v19.3.0

Update base image:

- RUN apt-get update && apt-get install -y apt-transport-https locales wget
- RUN apt-get install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common

Maven installation:

- RUN apt-get update && apt-get install maven -y

Docker installation for Docker-in-Docker testing setup:

- RUN apt-get install docker -y
- RUN curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
- RUN add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
- RUN apt-get update
- RUN apt-get install -y docker-ce docker-ce-cli containerd.io

Clair-scanner installation:

- RUN wget -q https://github.com/arminc/clair-scanner/releases/download/v8/clair-scanner_linux_386 -O /usr/local/bin/clair-scanner
- RUN chmod 0755 /usr/local/bin/clair-scanner

Sonar-canner installation:

- RUN curl -s -L https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-3.3.0.1492-linux.zip -o sonarscanner.zip \
- && unzip -qq sonarscanner.zip \
- && rm -rf sonarscanner.zip \
- && mv sonar-scanner-3.3.0.1492-linux sonar-scanner

Dependency-check installation:

- RUN curl -s -L https://dl.bintray.com/jeremy-long/owasp/dependency-check-5.3.0-release.zip -o dependency-check.zip \
- && unzip -qq dependency-check.zip \
- && rm -rf dependency-check.zip \

Java installation:

- RUN sudo add-apt-repository ppa:openjdk-r/ppa \
- && sudo apt-get update -q \
- && sudo apt-get install -y openjdk-11-jdk

Environment variables set for sonarscanner and Java:

- ENV SONAR_RUNNER_HOME=sonar-scanner
- ENV PATH $PATH:sonar-scanner/bin
- ENV JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
- ENV PATH $PATH:$JAVA_HOME/bin

Appendix 2. Docker images for testing environment

Docker version:

- Docker Engine v.19.03.8
- Docker Compose v.1.25.5

Docker images for testing material from Docker Hub:

- gitlab/gitlab-ee:latest

DVWA image for testing purposes from Docker Hub:

- vulnerables/web-dvwa

Docker images for testing tools from Docker Hub:

- gocd/gocd-server:v19.3.0
- sonarqube:latest
- arminc/clair-db:latest
- arminc/clair-local-scan:v2.0.6
- owasp/zap2docker-stable