Framework and Tools for IT Security within Logistics- and Infrastructure-oriented Operations

With a focus on Static Application Security Testing

Authors: Elias Seger, Fredrick Schedin

Supervisors: Joonas Pääkkönen

Examinator: Vijay Pratap Paidi

GIK28T – Thesis for bachelor degree in informatics

Thesis for Bachelor Degree in Informatics; 15 Credits; First Cycle [SWE: Examensarbete för filosofie kandidatexamen i Informatik, 15hp]

2022-06-08

Published in full text, freely available

Abstract

Static Application Security Testing Tools (SAST) is a security tool that claims to help with security in an IT system. Static Application Security Testing tools are technical solutions that operate within the continuous integration of the system. The tool uses frameworks such as OWASP and CWE to detect common vulnerabilities in the codebase by analysing code in the building and testing phase of continuous integration. The problem with SAST tools is that there are many different beliefs surrounding them. Some say they are crucial for security, while some believe they are less helpful and can even inhibit projects by introducing false positives. This thesis determines if SAST tools are an effective solution to security problems within in an IT system. The focus was on logistics- and infrastructure-oriented operations, which the partner company Triona operates within.

We use literature review to look at previously similarly conducted research combined with interviews with experienced people within the fields. This gives qualitative results that coupled with previous research can be generalized.

The results show that SAST tools are effective tools if used responsibly. Both the literature and interviews conclude that SAST tools are not enough on their own to satisfy the security requirements but must be combined with responsible use of the tools as well as code reviews and other types of testing. SAST tools are also shown to have some problems, mainly false positives, and false negatives. There are also problems related to the implementation of the tools. These problems are costs that comes with implementation, as well as the time spent on it. Other problems are bad communication with developer teams that led to developers not knowing what to do in case of errors shown by the tool. Interviews conducted provides information that SAST tools are not only tools for security but also helps with manageability of code bases.
Keywords: SAST, Continuous integration, SonarQube, OWASP, CWE, Security tools

Acknowledgements

We would like to express our gratitude's to the people who have helped us through this project. We wish to extend a special thanks to Triona for providing us with the opportunity to do our thesis, for being welcoming to us and providing us with assistance at every turn. Daniel Nilsson, our supervisor from Triona have helped us tremendously in shaping the course of the project and have given insights that have guided us through the work. We would like to thank the respondents at Triona who have taken their time and answering our questions. Finally, the assistance provided by Joonas Pääkkönen from Dalarna University was greatly appreciated.

Definitions

.NET	Open-source platform for developing applications in most platforms. Supports C#, F# and Visual Basic which are programming languages. **(Microsoft, 2022)** Used by partner company.
Static Application Security Testing (SAST)	Software that reviews source code of a project and looks for vulnerabilities and code smells (Oyetoyan, Milosheska, Grini, & Soares, 2018).
The Open Web Application Security Project (OWASP)	Non-profit organization that creates a list each year of the most important security vulnerabilities (OWASP, 2022).
Common Weakness Enumeration (CWE)	"Community-developed list of common software and hardware weakness types" (The MITRE corporation, 2021).
Integrated Development Environment (IDE)	IDE is a software application that combines different developer tools to write computer programs into a single graphical user interface (GUI) (Red Hat, 2019).
Software Vulnerabilities	Vulnerability that threatens the security of an application.
Manageability	Manageability of code measures the ease of access for newer developers. Manageability also makes it easier to maintain a codebase. It also means the code follows rules that makes the code somewhat universal for all developers on a project (National Instruments Corp., 2019).
Continuous Integration	"…a widely established development practice in software development industry, in which members of a team integrate and merge development work (e.g., code) frequently, for example multiple times per day." (Shahin, Babar, & Zhu, 2017)

Contents

1 Introduction

This chapter will contain the background of our study followed by problem description with the research questions. The aim of the study and the limitations is also mentioned. Lastly, a short description of our partner during this project is presented.

1.1 Background

When working in an IT environment, security is crucial. If IT security is lacking, there will be ripple effects throughout the whole organization and will potentially open the organization up to repercussions such as legal litigation or data leaks (Confessore, 2018). The IT vulnerabilities are hard to find, and it is easy to let vulnerabilities slip by while writing code without thinking about the consequences. To make matters easier, lists of potential security vulnerabilities have been created, which help with finding, prioritizing, and fixing these vulnerabilities. In this project we find the most important of these frameworks and find the respective tools that can help with the implementation of the security protocols that will help fix these issues.

The thesis is written with a partner company, which have projects within logistics- and infrastructure-oriented operations, and because of this there are possibilities of communicating with experienced people within the sectors. This gives opportunities for insights from real projects and people. Another focus will be to look at the issues through the lens of the .Net platform and to look at projects that involve these two. The reason for choosing this lens is that the projects investigated are using this platform. Through this work we investigate if SAST tools are a useful tool and to provide guidelines for future implementation of these tools.

Static Application Security Testing (SAST) are tools that help developers with writing secure, manageable, and readable code. The area is highly debatable where some studies like Lenarduzzi et al. (2020) claim the tools are not doing good whereas others like Yang et al. (2019) claim that it helps them tremendously.

The first commercial SAST tool was introduced in 1978 and in its inception the SAST tools were enforcing type rules more strictly than the programming language itself did. It checked if variables were used before the creation of it and checked for unreachable code that were impossible for the machine to reach because of bad flow control (Johnson, 1978). The sophistication of SAST tools have undeniably been increased. The second generation of SAST tools worked better because it could understand the difference between code and comments through a process called tokenization. The vulnerability database grew larger, and more languages were introduced. The third generation of security tools are the ones we use today. They cover technically every programming language and has huge datasets for vulnerabilities. There are also possibilities to filter out specific vulnerabilities if the developer would not want to fix the vulnerability.

This also helps with the problem of false positives since an already filtered out vulnerability would not show up again, hence removing the possibility of that specific false positive to impede the development. Another improvement are the dashboards that show the vulnerabilities decreasing (Lebanidze, 2008).

The reason why SAST tools are important is because it could help organizations that work with IT to negate or reduce issues related to common vulnerabilities. And since the SAST tools written about in this thesis use the frameworks with common vulnerabilities, the most likely vulnerabilities to exist are found and the developers can be notified and take precautions. This thesis investigates if the SAST tools has a role in IT security and what the benefits and drawbacks of using such a tool could be.

1.2 Problem description

IT Security is highly complicated and so incredibly important to all organizations that work with IT environments currently. Ideally no IT problems should impact an organization negatively. But as there are always incentives to attack and exploit organizations for financial gain, there will always be an unavoidable risk for every application in an IT environment. The solutions proposed to IT security are many, but in this thesis a specific approach is explored. This approach is Static Application Security testing.

The thesis is written to attempt to find out if SAST tools are an effective way of writing secure applications. The thesis is also exploring what the drawbacks as well as the benefits of using such an application could be.

From the description the following questions will be answered:

- Do Static Application Security Testing (SAST) tools help in writing secure code?
- What are the benefits and drawbacks of using a SAST tool?

1.3 Aim

The aim of the thesis is through increased understanding on how Static Application Security Testing works, and how it can affect IT systems and gauge how implementation of the tools can help solve IT security issues. Security frameworks are to be described to increase the understanding of how the tools work and why it is important to have the tool in mind when implementing security precautions in an IT environment. The aim is also to show what SAST tools are appropriate for the domains presented and describe how implementation can be simplified or improved.

1.4 Partner

Triona is a logistics- and infrastructure-oriented IT company with many different products and application in these areas.

The projects described below are the ones that the study has a focus on, since the interviews are conducted with experienced people within these projects. These projects are in the previously noted areas and can therefore be generalized to other IT systems within the same domains. These projects are also the projects we were given by the partner company to investigate and get information from.

1.4.1 Transport Network Engine (TNE)

Transport Network Engine is a support platform for applications and businesses to manage data about transport network. It is used in planning and monitoring of maintenance and operating activities. The configurability of TNE allows master data to be streamlined. The data collected can be used for many different parts of a business and allows for increased benefits and security of data. TNE also has operations for processing data and generating it, which can be used to get data about various parts of a road network, for example (Triona, 2022).

1.4.2 Tracs Flow

Tracs Flow is an order management system that allows for processes in the transport and construction sector to be simplified. It has plenty of features relating to contracts, resource management, order tracking and many more. The application gives an operation overview of the business. There are both web and mobile application to manage the processes (Triona, 2022).

1.4.3 Graphical Development Tool

Graphical Development Tool is a development tool which generates code by using graphical symbols instead of using source code. Behind every symbol there is a part of code that will run, and the output of several symbols will generate a working application for engineers to use. It means that the engineers using the application do not need to know how to code (1, 2022).

1.5 Scope and limitations

The study has several delimitations, and they are:

- Focusing on the domains of which our partner organization resides
- Focusing on Static Application Security Testing
- Focusing on the three projects described in the next section

The scope in turn becomes SAST tools within Triona's domains. This was done to ensure time existed to interview our subjects in time and to get an overlook of the three stated projects, and the differences. There are other ways that the work could be done, those are stated in section 6.

2 Literature review

This chapter presents examined literature, introduces the concepts, and tools that were studied to achieve the purpose of the study, which is to answer our research questions.

2.1 Static Application Security Testing (SAST)

Static application security testing is a way of testing an application for software vulnerabilities. It does it by finding vulnerability within code and does not require the application to be running. SAST is what is called white box testing which means it only looks at the code of the program and not at program while it is running. The SAST tools uses lists of known vulnerabilities to scan the source code and warns the developer if any known causes for alert are present. SAST is implemented when continuous delivery is a key to the development process (Oyetoyan, Milosheska, Grini, & Soares, 2018).

2.2 Static Application Security Testing effectivity

In a study where SAST tools were compared to each other, the effectiveness was measured by how many of the common loopholes were found. From the results it could be found how many false negatives and false positives were found. False positives refer to cases where vulnerabilities were found, but none existed in the code, whereas false negatives refer to vulnerabilities that existed, but were not found (Li & Cui, 2010).

Another idea found in Li and Cui (2010) was that the effectiveness of software vulnerability detection could be improved by using different tools, mainly dynamic testing. Dynamic testing is when the testing is done on the running application and not looking at the codebase. It removes the abstraction in only watching the code and evaluates the application in the same way a user would interact with the program. It was also conjectured that pairing the two different approaches would reduce the false negatives, thus avoiding additional vulnerabilities to go unnoticed.

2.3 Code Smells

Code smells are described in van Emden and Moonen (2002) as "…code that indicate that refactoring can be applied." Refactoring in this case is described as improving the internal structure of a software system but the external behaviour has not been changed. The paper clarifies by saying that if code smells are fixed, "It improves the design of a software system after it was written by tidying up code and reducing its complexity."

SAST tools also help by analysing code smells. In Sjøberg et al. (2013) the authors attempted to quantify the effects of code smells on maintenance work. And it was concluded by the study that code smells did not correlate with more effort on the developer's part when maintaining the code. In Palomba et al. (2018) it was found that

smelly classes (classes with code smells) had a higher change and fault-proneness than smell-free classes.

2.4 Challenges in using Static Application Security Testing

The challenges with SAST are discussed in Yang et al. (2019) where the authors lay out where the issues with SAST exist. The first issue discussed is the fact that SAST presents the developer with a ton of security issues but does not help with prioritizing the issues. Another issue with SAST are the false positives, where security vulnerabilities are reported where it is later found out that none exist. This leads to time being wasted to investigate issues that are not a problem. This is a problem since time goes to troubleshooting code that has no issues when time could instead be spent fixing real vulnerabilities or development. Another problem discussed is that the vulnerabilities are presented without actual solutions to the problems. There are usually generic solutions, but nothing concrete that helps developers with the specific vulnerability at hand (Yang et al., 2019).

Other issues presented in Lenarduzzi et al. (2020) which looks at the specific SAST tool SonarQube is that SonarQube violations on its own might not be prone to faults and should not be considered as bugs. They suggest other tools with machine learning to improve SonarQube to customize a solution for every company to have a model that is specific to the company. The report also explains that violations classified as bugs do not seem to cause faults for the software. It also suggests that violation severity, which is a classification in SonarQube that attempts to predict how severe a vulnerability might be to the software, is not related to fault-proneness. This is evidence that software developers should not always look at SonarQube's own severity level to determine what vulnerabilities should be prioritized.

Marcilio et al. (2019) discusses the number of issues and how low the solution rate was in a project. It was only 13 % and this might speak to how many of the SonarQube issues are irrelevant and not important to fix. It might also suggest that many of the projects analysed does not have great security and could have vulnerabilities that are not fixed. The same paper also showed that 20 % of the total rules in the SonarQube ruleset are 80 % of the fixed issues within these projects, which might insinuate that the most critical issues are fixed, but the developers realize which ones are not important to fix and they are left out of the maintenance since they are deemed by the developer to be a "safe" vulnerability for their system. The rules and ruleset in this case refers to the rules that are applied to find vulnerabilities in the code base. Examples of these rules can be found in sections 2.5.2 and 2.6.2. This data could also help future implementations to discard the rules that are not necessary to fix and prioritize those who often are (Marcilio, et al., 2019).

The 80/20 principle is a known phenomenon that applies to many areas. An example of the principle was shown in 2003 by Microsoft,

where they analysed their bugs and realized that 80 % of the errors were caused by 20 % of their bugs (Rooney, 2002).

2.5 OWASP Security factors

OWASP is a document used for developers when trying to find security weaknesses within their projects. It is a document that represents a consensus on data surrounding security weaknesses. The OWASP document consists of ten vulnerabilities which are sourced from two diverse types of data. Two of the vulnerabilities are contributed from a survey done by professionals within the software development field, and the other eight are extracted from data gathered from organizations. From the two vulnerabilities that are drawn from the survey, OWASP selects their initial security weaknesses that they gather from looking at the CWE list and what vulnerabilities are close to making it to the CWE Top 25. They then vet their initial choices and determine a list for the survey to check if something is missing. They then publish their survey to industry professionals, which helps by ranking the proposed vulnerabilities in a top four list. This is then taken and analysed, and two risk are contributed to the total top ten list of vulnerabilities. This allows vulnerabilities that are not represented in data to be contributed by practitioners, and they vote on what they think are some the more important and high-risk vulnerabilities (OWASP, 2022).

OWASP has two group of factors that impact the severity of a security vulnerabilities. One of the group of factors estimate the likelihood of a vulnerability to happen. The other deals with the impact if the vulnerability were to happen. These two groups of factors sum to a severity rating that measure how severe a security vulnerability is.

Threat agent factors are factors related to the agent that imposes a threat to the IT environment. The factors that measure the threat agent are Skill Level, Motive, Opportunity, and Size.

Vulnerability factors are the factors that impact the vulnerability of the system. These factors estimate likelihood of a security vulnerability impacting the system. These measurements are Ease of Discovery, Ease of exploit, Awareness, and Intrusion Detection.

Technical impact factors are factors that measure the magnitude of impact if a security vulnerability were to be exploited. These factors are Loss of Confidentiality, Loss of Integrity, Loss of Availability and Loss of Accountability.

Business impact factors are affected by technical impact factors but are more about the company itself. The factors are common areas of business that are impacted by security collapses. These factors are financial damage, Reputation damage, Non-compliance, and Privacy violation.

The severity of the risk depends on the forementioned two or four groups of factors. The severity of each factor is what creates a severity measurement that is the average of each factor´s value.

2.5.1 OWASP 3 Examples

OWASP as previously explained by the report is a list of the Top 10 security weaknesses. To give some context on what is reported in these lists we show some examples of what they could look like, and how to solve the issues.

Broken Access Control or A01:2021

Broken Access Control is a failure in enforcing what users can access what data. This problem leads to people not being able to access what they should, or worse, people who should not get access to important data gets access to it (OWASP, 2022).

Some solutions for A01:2021 can be found below:

- Deny by default, meaning that all access to everything should be denied unless specifically configured to be accessible.
- Log failures of access control, alert admins if problems occur.
- Limit APIs to only accept a specified number of requests per a specified time frame.

Cryptographic failures or A02:2021

Cryptographic failure is a broad term that relates to a failure of protecting data with reliable cryptography. This OWASP vulnerability contains three CWE vulnerabilities, these are “Use of Hard-coded Password,” “Broken or Risky Crypto Algorithm” and “Insufficient Entropy.” Use of Hard-coded Password are an issue that means password are hardcoded for each instance of a program. This means that every user that uses the application has the same password, and it cannot be changed. Broken or Risky Crypto Algorithm refers to using bad cryptography. Cryptography is there to ensure passwords are safe, and cryptography that once was good and safe might today not be safe, since technology advances. Insufficient entropy refers to an issue where values that are generated are not predictably generated and could be guessed by understanding how the generation of the value occurs. These issues are related to matters like sending data in clear text, old cryptographic algorithms, bad server certificates, having crypto keys in source code repositories and other similar issues (OWASP, 2022).

Some solutions for A02:2021 can be found below:

- Identify what data should be protected by checking laws, requirements, and regulations.
- Use up-to-date encryption.
- Encrypt all sensitive data

Injection or A03:2021

Injection is a vulnerability that talks about injecting malicious code into fields that are used for user queries or similar. It can happen when SQL queries are not cleaned before being sent to the database, which could cause the database to be injected with malicious code that could erase a table, send back whole tables, or send user tables with hashed passwords, for example. An example provided is if the malicious user uses a query value to get an account view but can instead of only sending the user id send in an expression such as "or '1' = 1" which causes the whole table to be sent back. The example provided is shown below for clarity (OWASP, 2022).

```
http://example.com/app/accountView?id=' or '1'='1
```

Some solutions for A03:2021 can be found below:

- Use a safe API. It avoids the interpreter.
- Use escape syntax for the interpreter if API is not used.
- Use SQL controls (Such as LIMIT) to prevent mass requesting of documents.

2.6 Common Weakness Enumeration (CWE)

Common Weakness Enumeration, also known as CWE is a list that is community-developed and contains software and hardware weakness types. Weaknesses in this context are bugs, faults, flaws, and other errors that occur in software or hardware implementation, code, or architecture. If the weaknesses are left unaddressed the systems or networks could be vulnerable and encounter attacks that can be harmful for an organization (The MITRE corporation, 2021). The CWE list serves as a language that can be used to describe and identify weaknesses in terms of CWEs.

The main goal of the CWE list is to stop the vulnerabilities at the source by educate software and hardware architects, programmers, and designers, which means that it targets both developers and security practitioners. The CWE list will help to eliminate the most common mistakes and prevent security vulnerabilities before products are delivered, which has plagued the software and hardware industries (The MITRE corporation, 2021).

According to The MITRE corporation (2021), CWE helps both developers and security practitioners to:

- Describe and discuss software and hardware weaknesses in a common language.
- Check for weaknesses in existing software and hardware products.
- Evaluate coverage of tools targeting theses weaknesses.
- Leverage a common baseline standard for weakness identification, mitigation, and prevention efforts.
- Prevent software and hardware vulnerabilities prior to deployment.

2.6.1 CWE Top 25

The 2021 CWE Top 25 is a list of the most common and impactful weaknesses suffered over the previous two calendar years (The MITRE corporation, 2021). These weaknesses are dangerous because they are easy to find, exploit, allowing adversaries to steal data, prevent an application from working and even take over systems completely (The MITRE corporation, 2021). The CWE Top 25 list is a useful source that helps all types of roles within an IT development environment with insight into the most severe and present security weaknesses.

To create the Top 25 list, the team uses Common Vulnerabilities and Exposure (CVE) data from a database called National Vulnerability Database (NVD) and a scoring system called Common Vulnerability Scoring System (CVSS). The scores are correlated with each CVE record and a formula is applied to the data to get a specific score according to weakness. The score is based on prevalence and severity. The database NVD and the scoring system CVSS can be found within National Institute of Standards and Technology (NIST). All the terms used here are explained below (National Institute of Standards and Technology, 2022).

National Vulnerability Database (NVD) is a U.S government repository which is based on vulnerability management data that is represented by using the Security Content Automation Protocol (SCAP). SCAP is a method for using standards that are specific to allow automated vulnerability, measurement, and policy compliance (National Institute of Standards and Technology, 2022).

Common Vulnerability Scoring System (CVSS) is an open framework for communicating the severity and the characteristics of software vulnerabilities. The scoring system consists of three metric groups, and they are: Base, Temporal, and Environment. The Base generates a score between one and ten, which then can be modified when scoring the Temporal and Environmental. CVSS is a well-suited standard measurement system for organizations and governments that have the need to measure the vulnerability (National institute of standards and technology, 2022).

Common vulnerabilities and exposure (CVE) mission is to “Identify, define, and catalogue publicly disclosed cybersecurity vulnerabilities.” The weaknesses are calculated by its frequency of occurrence and its severity which are both normalized values. The frequency is determined by how many times a CWE is mapped to a CVE in the NVD. This approach only allows CWEs with an associated CVE to be calculated and scored. The following formulas determines the rankings of the CWEs within the CWE Top 25 list (The MITRE corporation, 2022).

First, we count all the CWE vulnerabilities that is found in the NVD in Formula 1 and then Formula 2 calculates the frequency of the

mapped vulnerability in the NVD. Formula 3 then calculates the severity of the specific vulnerability by taking the average CVSS score and mapping it to the CWE vulnerability. Formula 4 calculates the danger presented which is a combination of frequency and severity.

The formulas below come from OWASP (2022).

Formula 1 (Counting CWE mapped in NVD):

$$Freq = \{count(CWE\ X' \in NVD)\ for\ each\ CWE\ X' in\ NVD\}$$

Formula 2 (Finding the frequency of the mapped CWEs):

$$Fr(CWE\ X) = \frac{(count(CWE\ X \in \text{NVD}) - \min(\text{Freq}))}{(\max(Freq) - \min(Freq))}$$

Formula 3 (Calculating the severity):

$$SV(CWE\ X) = \frac{(average\ CVSS\ for\ CWE\ X - \min(CVSS))}{(\max(CVSS) - \min(CVSS))}$$

Formula 4 (Calculating the danger):

$$Score(CWE\ X) = Fr(CWE\ X) * Sv(CWE\ X) * 100$$

2.6.2 CWE 3 examples

CWE as previously explained in section 2.6.1 is a list of the Top 25 security weaknesses. To give some context on what are reported in these lists we will show some examples of what they could look like, and how to solve the issues.

Out-of-bounds Write or CWE-787

Out-of-bounds write is what happens when software writes data past the end of the file and does not exit. An example given by The MITRE Corporation (2021) is if a sequence of integers is created that has an index of three. Then an attempt can be made to insert a fourth index of the sequence, which does not exist, as the sequence only goes to the third index. The assignment is out of bounds.

Some of the solutions provided by The MITRE Corporation (2021) for CWE-787 were:

- Using a language that does not allow these types of weaknesses. Java has its own memory management inherently prohibits this behaviour.
- Using compilers and extensions that provide buffer overflow detection.

- Compile the software to randomly arrange the program's memory. This makes reliable exploiting this weakness for a specific purpose close to impossible.

Improper Neutralization of Input During Web Page Generation ('Cross-site scripting') or CWE-79

Cross-site scripting or XSS is when a web request with data that the browser executes is inserted to a web page and not neutralized before entering the page. And through this, the malicious data persists, and when another user enters the same page, the script is run, and the user is affected. An example presented by CWE is when a GET request executes by a user, a parameter for the URL is arbitrary and can therefore be changed by a malicious actor to contain scripts. So instead of the URL parameter containing the username of the user, it can be changed to contain JavaScript that in the provided example contains HTML that leads the user to try and log in, but the user information is sent to a server the malicious actor is providing instead of to the trusted website. Examples provided in the documentation also explain how the URL parameters does not have to look like "www.trustedsite.com/hack" but can be obfuscated to an extreme length so that there is no way a regular user would suspect the URL parameters has been changed at all (The MITRE Corporation, 2021).

Some of the solutions provided by (The MITRE Corporation, 2021) for CWE-79 were:

- Assuming all inputs done by users are malicious. Have a list of inputs that are allowed, and the rest are prohibited.
- Using mechanisms that are separating the data and code. These mechanisms could provide encoding and validation automatically.
- Understanding and implement different encodings depending on input and output.

Out-of-bounds Read or CWE-125

Out-of-bounds Read occurs when code tries to read past the end or before the beginning of data. This may cause crashes to the program, or it could allow a malicious actor to read data from previous memory locations and get sensitive data. An example provided is that negative values can be inputted to get an index of an array, and in extent access to memory outside of the index can be made (Common Weakness Enumeration, 2022).

Some of the solutions provided by The MITRE Corporation (2021) for CWE-125 were:

- Using programming languages that provide its own abstractions of memory.
- Assume all input from users are done maliciously. Consider all vectors of attack, length of input, type of input. An example provided is that the input "Boat" should not be accepted if

the expected input are colours (Common Weakness Enumeration, 2022).

2.7 .Net Platform

.Net is an open-source development platform that can be used for many different types of applications such as web applications and desktop applications within many different operative systems. The .Net platform includes several implementations. From the .Net framework which is the original implementation that only runs on Windows, to the .Net 5+ that runs on any platform. The projects we will investigate in our studies are limited to the .Net platform.

2.8 Continuous integration (CI) systems

This section explains what continuous integration is and provides two examples of environments with continuous integration.

2.8.1 Azure DevOps

Azure DevOps is a framework used for development and operations surrounding development. It makes collaboration easier and aims to make productivity and teamwork easier. DevOps has four stages which is the Plan phase, Develop Phase, Deliver Phase and Operate Phase. In the Plan phase projects are defined and plans all parts of a project. In the Develop phase, the coding is key. Writing, testing, reviewing, and integrating the code to the codebase is the main part of this phase. The Deliver phase is about delivering the product to the customer. This is where the Continuous integration (CI) and Continuous Delivery (CD) comes in and this is an important part of our research. CI will be explained further below. The Operate phase is about maintaining the delivered product (Microsoft, 2021).

Continuous integration is an important part of many DevOps projects. It aims to automate the process of testing the code which is committed by all the developers on a project. Each time code is attempted to be inserted into the main codebase, a process begins, which usually has many tests that checks if the code is OK. This is where security vulnerability tools run their scans. This should be integrated easily with the tools that have integration compatibilities with the CI systems. In Appendix C: Static Application Security Testing Tools compatible tools are presented (Microsoft, 2021).

2.8.2 JetBrains TeamCity

JetBrains TeamCity is an easy-to-use continuous integration (CI) server for developers, admins and build engineers. It can detect changes in repositories and trigger builds whenever new code is checked in. TeamCity can be designed to perform the build activities, including the compilation of source code, running unit and integration tests. The design of TeamCity will help users to follow best practice of CI, and follow the approach build once and deploy everywhere (Mahalingam, 2014).

2.9 Tools for security implementations

In this section we give a brief introduction to security tools. In Appendix C: there are comparisons between the tools.

2.9.1 SonarQube

SonarQube is described as an "automatic code review tool to detect bugs, vulnerabilities, and code smells in your code." In practicality this means it is a tool that checks code as it is being built, and it checks it for known vulnerabilities in the CWE and OWASP top lists. What makes SonarQube different from many other similar tools is that it has an open source and completely free community edition. SonarQube also has a tool that checks code as it is being written, it called SonarLint. Together with this, a continuous integration is possible, which does a more thorough scan of the whole codebase. This generates a report which is sent to developers which can then resolve the issues or decide if the vulnerability is an actual problem. A SonarQube scan can also be added each time a pull request is done (SonarQube, 2022).

2.9.2 Static Reviewer

Static reviewer is a part of Security Reviewer and serves to scan the code for vulnerabilities. It is OWASP and CWE compliant. Static reviewer can be used on multiple programming languages at once. It can also be used as a plugin for Azure DevOps and can then be run with the continuous integration DevOps does. It does need a Jenkins or Bamboo server to run the Secure Code Analysis, which is a drawback, as other security implementations do not require this (Security Reviewer, 2022).

2.9.3 Kiuwan

Kiuwan (2020) explains their goals as being: "

- Detect security vulnerabilities as early as possible in the development life cycle.
- Reduce issues – bugs – in the technical aspects of applications: performance, efficiency.
- Manage costs associated with development and maintenance, from internal and external resources.
- Align developed applications with business goals and missions.
- Increase team productivity.
- Gain greater control – governance – of application development and maintenance outsourcing. "

Kiuwan can achieve this with local analysers, cloud analysers but more importantly Kiuwan can be integrated with TeamCity. This allows for Kiuwan's analysers to work on the code within the building of projects. A build trigger can be defined where Kiuwan runs when changes on the codebase are detected, for example. After this has been done, Kiuwan has a dashboard where the result is displayed. In

the dashboard vulnerabilities and risks can be viewed (Kiuwan, 2020).

2.9.4 Parasoft dotTEST

dotTEST is an integrated development testing solution for Microsoft .Net framework and can be used with a user's build tool and continuous integration (CI) infrastructure. The integrations allow the user to automate a wide range of best practices from CWE and OWASP to help the code be more secure and the user can make reports from the analysis of the code that can help the company make better decisions in the future. The user also sends information to the Parasoft Development Testing Platform (DTP). DTP analyses the collected data and transforms it into actionable findings which will help the user in the process of improving the code. So, while the user uses the tool, they will continuously improve it (PARASOFT, 2019).

The dotTEST engine can also monitor and collect data during unit testing or functional testing on running applications, and that specific information is sent to DTP to help the user to assess the quality of the tests (PARASOFT, 2019).

2.9.5 PVS-Studio

PVS-Studio is a static analyse tool that works with C#. PVS-Studio can be used both as plugin but can also be integrated in the Continuous Integration of both TeamCity as well as DevOps. PVS-Studio can be integrated with SonarQube as well so that profiles with warnings can be created and viewed. This gives a bigger scope of vulnerabilities that can be found, which in turn should help find more vulnerabilities (PVS-Studio, 2022).

2.10 Integrated Development Environment (IDE)

IDE is a software application that combines different developer tools to write computer programs into a single graphical user interface (GUI). By combining activities such as, editing source code, building executables and debugging will make the programmer more effective. The source code editor in an IDE will facilitate the developer by syntax highlighting, provide auto-completion for specific languages, and checking for bugs whilst coding (Red Hat, 2019).

Local build automation helps to automate operations such as converting computer source code into binary code, packaging binary code, and running automated tests as part of building a local version of the software for usage by the developer. Debugging is also a part of the IDE which allow programmers to examine different solutions and inspect their code when the program does not run correctly (Red Hat, 2019).

2.10.1 How it works (IDE)

Both static analysis such as SonarQube, and dynamic analysis tools such as ZAP have been developed to detect security flaws in code. These tools come with their own GUI to display analysis from the

code that have been analysed which will require the developer to go back and forth with the coding environments, and tools for analysing security problems. Here is where the IDE plugins came in handy, to

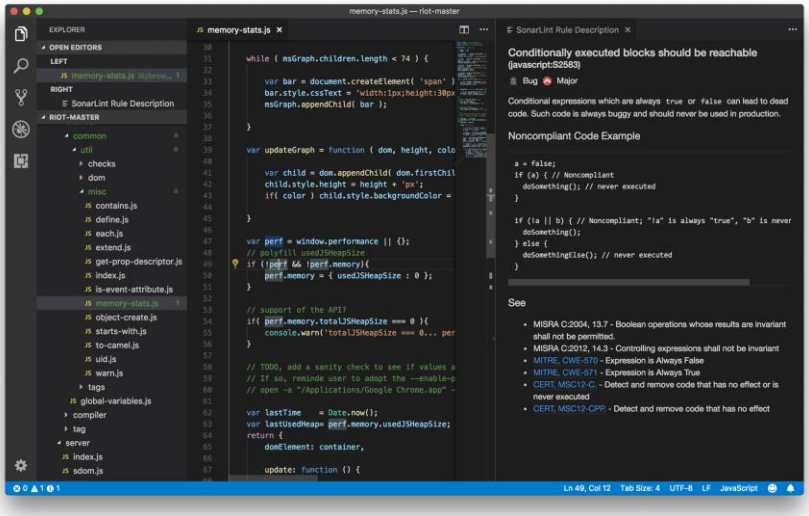

Figure 1 SonarLint Example

provide a more seamless experience for the developer. This allows the developer to check the code for flaws within the IDE and make the detection of the specific flaw in a much earlier state of the software development. An example is provided in Figure 1.

Much like regular compiler warnings and errors, the plugin works the same according to visualization. The identified flaws are displayed in an informational pane as well as the problematic code lines with markers in the code editor pane. This type of visualization is quite familiar to developers which will allow them to make changes in the code while viewing the result. There are some differences among the plugins according to analysis reporting. Some plugins are more detailed in their visualization then others. The more detailed plugins reveal for examples attacks, how the faulty code can lead to specific attacks, vulnerable and secure code, and risk ratings whilst other plugins only provide brief descriptions of the fault or even just the name of it. Some plugins also suggest mitigation strategies. Usually, the detailed reporting techniques are more targeted to educate the developer about the specific attack and do not serve as a quick fix to the mentioned problem. The developer can turn off some warnings and select/unselect specific vulnerability checks, to make the plugin more flexible. Prior research suggests that these types of functions make the plugin more usable (Baset & Denning, 2017).

2.10.2 SonarLint

SonarLint is an Open-Source extension that helps developer get real time analysis of the code they are writing. Its purpose is to give feedback on mistakes in the code such as bugs and vulnerabilities. It also gives suggestions on how the written mistakes can be fixed (SonarLint, 2022).

2.10.3 Intellisense

Visual Studio Code IntelliSense is a code completion tool that helps developers by giving suggestions while writing their code. The tool can give several types of information. List members give suggestions when a specific trigger is typed. In most languages the trigger character is a (.). It helps by finding methods within a class for example. If there is a class called Car, and that Car class has a method called Drive, when writing "Car." suggestions will be shown for the drive method. Another type of suggestion is "Parameter Info" which gives information on what parameters is required for a method that is called. If the Car class is used to call the Drive method and the Drive method requires a speed to be called, the parameter info will show that an integer value is required as a method in the Drive method. Quick info will show how the code declaration is wrong and how it can be corrected. If Car.Drive("Hello") is written the Quick info message will show that the parameter is wrong and how to fix it (Microsoft, 2022).

2.10.4 ReSharper

ReSharper is a code quality analyser that runs while a developer is writing their code. It serves to help the developer fix problems with their code in real time. ReSharper differs from Intellisense and SonarLint in that Resharper have instant fixes and refactoring together with code generation. This makes ReSharper more powerful, but also more difficult and dangerous to use (JetBrains, 2022).

2.11 Build phase (Continuous integration)

In Shahin et al. (2017) continuous integration is described as "a widely established development practice in software development industry, in which members of a team integrate and merge development work (e.g., code) frequently, for example multiple times per day." The continuous integration is also described to automate building and testing of the software.

It may seem like an IDE solution to the problem might work where it analyses the code on the individual developer's computer, but Bolduc (2016) explains that a continuous integration may work better since there are some key issues with IDE or desktop analysis solutions. The issues in Bolduc (2016) are described as:

- The code available on the developer's machine may be a subset of the real code that will be used in a full-fledged build of the system. This is common when using componentization of the codebase. In this case, the desktop analyser might miss links between components and either miss defects or falsely report defects that are not present (this is called a false positive).
- It is more difficult to ensure that developers are consistently using the analysis tool and fix the issues raised.
- It is difficult to track and manage the defects on individual developers' machines (Bolduc, p. 37).

Continuous integrations are used to counteract these issues. Continuous integrations also have the goal of having frequent builds, ideally every commit would be analysed, but this is not realistic as bigger codebases could take hours to be analysed and therefore analysis is often run nightly to omit as much downtime as possible.

Bolduc (2016) also talks about the difficulties when implementing a SAST tool into continuous integration. First, they discussed the vast amount of data provided after each CI run. They solved this by implementing faster CI builds during daytime that only shows defects and having nightly builds that goes more in depth. An issue with this was how they would recover when defects were found. And they solve this by omitting past defects from future builds so that the build still runs without faults. This helps developers to find the faults and have time to fix them, but also lets future builds run without past failures getting in the way.

Education is also something that was discussed as a problem. There were technical details about the tool that made it hard for developers to understand what their work was when a defect was found. And this was solved by clear communication and future engagement with the team when new changes occur.

3 Method

This section describes the research process and goes through all the steps in the collection of data as well as the analysis. It goes through the interviews as well as the literature review methods we used. A discussion on ethical considerations can also be found in the later section.

3.1 Research process

In this thesis the research process is inspired by Oates (2005) as shown in Figure 2. The yellow boxes are the approach of how to perform the project.

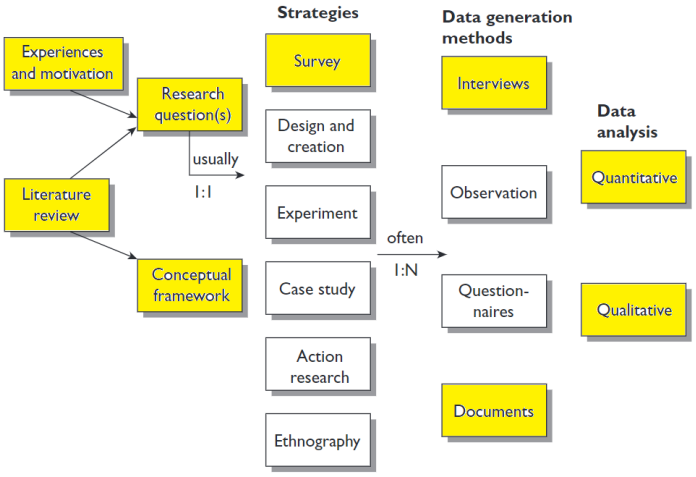

Figure 2 Model of the research process (Oates, 2005)

There are a few things to keep in mind before moving on to the choice of strategy and that is to think about the motivation as well as subjective experiences. By gaining clarity in those two areas and reading through the literature, research questions should be easier to address how to make a conceptual framework, which explains how to organize thoughts regarding the topic and methodology.

The chosen strategy is a survey, and according to Oates (2005), a survey is a way to obtain same kinds of data from a larger amount of people, in a systematic way. For the data generation method interviews and documents have been selected. The reason behind the data generation methods is to get a deeper insight and a greater knowledge in the subject. The qualitative data analysis helps answer the thesis research questions and fulfils the purpose of the report. The research question “Do Static Application Security Testing (SAST) tools help in writing secure code?” will be answered by obtaining knowledge through the literature and by the semi-structured interviews which will help to gather information from employees with different experiences to get a more general understanding. The interview part will be further explained in section 3.4.1. The research question “What are the benefits and drawbacks of using a SAST tool?” will also be answered through the literature. The semi-structured interviews that are about this question will be answered by employees

within Triona that have some experience of SAST tools, either in projects at Triona or other projects, in order to gain their perspective on the subject and their experiences, which is the purpose of the interviews.

3.2 Literature review

The purpose of the literature review according to Oates (2005) can normally be divided into two distinct parts. The first part of the review is where the student search information to get a research idea and trying to find relevant material for a specific topic. This part will help the student to form a research problem. When a topic has been chosen and a research question has been made, the second part of the literature review begins and carries on throughout the project. The aim of this part is to bring forth evidence that the project has provided new knowledge to the studied topic (Oates, 2005).

A literature review can be broken down into seven distinct parts of how to conduct it, the steps are: searching, obtaining, assessing, reading, critically evaluating, and writing a critical review (Oates, 2005). A literature review can be broken down into seven distinct parts of how to conduct it, the steps are: searching, obtaining, assessing, reading, critically evaluating, and writing a critical review (Oates, 2005). During the literature review OWASP, CWE Top 25, security tools IT-system, SAST, .NET and continuous integration were search words that was gathered through discussion on what data and information we would need. Google scholar, Summon, and Google.com were used when searching for the specific key words. Appendix A includes a table with search words and what search engines were used to search and obtain information.

To assess if the articles were relevant for the specific topic the headings and the abstract were read to get an idea about the content of the articles. If it matched what was searched for, the entire article was read through, or at least the parts needed. Some articles generated other or new articles when relevant referring was found.

3.3 Strategies

According to Oates (2005) a strategy is an approach used to answer the research questions, and the strategy that was chosen for this project were survey.

“A Survey focus on obtaining the same kinds of data from a large group of people (or events), in a standardized and systematic way. You then look for patterns in the data using statistics so that you can generalize to a larger population than the group you targeted” (Oates, 2005).

3.4 Data generation method

By using a survey as the strategy, it is often common to be assumed that questionnaires are used as the data generation method. But that

is not always the case. Oates (2005) mentions that interviews, documents, and observation are also something that can be used. The methods that are used in this project to gather data are interviews and documents.

3.4.1 Interviews

The interviews will be held with employees at Triona, and the respondents have different experiences and works in various projects with distinct roles within the said subject to make it as relevant as possible. This will help to get information on what security framework and tools the employees use within different projects, but also what their personal thoughts are about them.

The reason interviews are chosen is because interviews will help by obtaining detailed information about specific cases or applications within a company instead of only getting the data through published literature (Oates, 2005). Also, what security issues the respondents feel are most important or neglected in the company and should be focused on more. The interviews are semi-structured which is meant to keep the respondent on topic, but also let them expand their topics and elaborate where they might feel it is important and it also makes it possible for follow-up questions (Oates, 2005).

The interviews will be conducted at Trionas premises and will be recorded to make it easier for the authors to transcribe each interview. The interview questions can be found in Appendix B as a template and the questions are customized to each respondent according to their knowledge and experience. The follow-up questions that arose during the interview are not scripted in Appendix B because they are considered more as discussions. The transcripts are not in the appendix to comply with GDPR.

3.4.2 Respondents

Respondent 1

Respondent 1 works as a Senior Developer at Triona, but also states that they are working as an IT architect on some projects. They have been working with IT since 2000. Now works at a project that develops a Graphical Development Tool.

Respondent 2

Respondent 2 works as a Chief Technical Officer at Triona, but has a background as a system developer, lead developer and as an IT architect.

Respondent 3

Respondent 3 works as a lead developer, at a project called TNE, and is also a tool specialist. The have been working with IT for eight years. First three years as a system developer and last five years as a lead developer.

Respondent 4

Respondent 4 has worked at Triona since 2019, directly after their graduation. Started as a system developer but is now a lead developer at a project called Tracs Flow.

Respondent 5

Respondent 5 has worked with IT since 1986, and at Triona since 2001. Works as an IT architect at a project that develops a Graphical Development Tool.

Respondent 6

Respondent 6 has worked with system development since 1994. Works at a project called Tracs Flow and has been there for three years.

Respondent 7

Respondent 7 works as a Senior Developer at Triona. They have been working in the current role for five years. Now works at a project that develops a Graphical Development Tool. Has previously worked with business systems.

3.4.3 Ethical Considerations

Ethics are very important when conducting interviews. The research must be conducted with both ethical and legal considerations. When dealing with people in research, and more specifically when doing a survey, the people involved in answering questions are called respondents. These are the people to take into consideration when doing research (Oates, 2005).

Oates (2005) defines the rights of participants as “

- Right not to participate
- Right to withdraw
- Right to give informed consent
- Right to anonymity
- Right to confidentiality” (Oates, p. 56)

Oates (2005) describes these rights more in depth as written below.

- The right not to participate means the respondent can say no to participate in the research. And this means a researcher should not continually ask the respondent for an interview once they have said no once.
- The right to withdraw means the respondent can withdraw their statements at any time. This means the respondent can change their mind at any time, even if the research in turn becomes useless.
- Right to give informed consent means the respondent knows what they are agreeing to before giving consent to an interview. They should know why the research is being done, and how long the interviews will be for example.

- Right to anonymity means the respondent has the right to not be mentioned by identifiers. This means we change their names to Respondents 1 through 7. This thesis also does not gender the respondents.
- Right to confidentiality means the respondents information is confidential and should not be leaked. This should be accounted for in our actions and we should not leave documents open on our PC and leave the PC unlocked when we leave it, for example. This also bears with it responsibility surrounding the information on partner companies. So that information that could be used by competitors to gain the upper hand over the partner company is not included (Oates, 2005).

3.4.4 Documents

Oates (2005) says that documents are another source of data as an alternative to interviews, which can be divided into two types: found documents and researcher-generated documents.

The approach used in this project was found documents which means that the documents already exist, such as documents found in organizations. It also involves publications like academic literature, for example, journal articles and conference papers. Oates (2005) explains that previous research also provides data that can be reused in projects. This type of data is often called secondary data and is an approach that has been used during this project.

By using this data generation method together with interviews, we might find generalizable data that will prove useful in the field of IT security and can help as a complement and support the statements made in the interviews. It will also give us a background and place our conducted research in a context with the already published research (Oates, 2005).

3.5 Data analysis

"The idea of data analysis is to look for patterns in the data and draw conclusions." (Oates, 2005)

Oates (2005) presents two types of analysis of data. It can either be a quantitative data analysis, which means that the data are based on numbers and are often used when gathering data from experiments and surveys. The other type is a qualitative data analysis and includes words, sounds, and images. These parameters are often found in interviews, tapes, company documents, and websites.

In this project, both qualitative data from interviews as well as literature that is based on a case study have been used. But also, quantitative data have been used where some of the sources describe the numbers and present sets of data that describes certain situations. By reading the transcribed interviews, the paper attempts to create a generalizable picture of what developers think about the different

topics, and when the two are paired, an attempt was made to strengthen or oppose the views expressed. This in turn creates data that both takes from the experience of individual developers but also takes quantitative data that explain reality.

The authors initially analysed the data on their own to form their own impressions of the findings based on the information gathered from the interviews and the literature. The authors' results were then compared and discussed to obtain the most accurate final result and also to make it as credible as possible.

The authors strategy of analysing the acquired data separately was addressed at an earlier stage, and it was decided to do it in a similar way. Oates (2005) gives examples of three key themes to identify relevant information from the collected data which the authors adopted, which were:

- Segments that bear no relation to your overall research purpose so are not needed.
- Segments that provide general descriptive information that you will need in order to describe the research context for your readers.
- Segments that appear to be relevant to your research questions.

When the data were divided in these particular segments, it became easier for the authors to obtain relevant information about the research questions and to bring forth a final result.

4 Results

In this chapter our results are presented from both the literature review and interviews. Sections 4.2 – 4.4 addresses the research question "Do Static Application Security Testing (SAST) tools help in writing secure code?" by summarizing the interviews and comparing them to the literature. Sections 4.5 and 4.6 answers the question "What are the benefits and drawbacks of using a SAST tool?"

4.1 Static Application Security Testing (SAST)

In the literature as well as the interviews there was evidence suggesting that Static application security testing is an effective way of finding bugs and vulnerabilities in a code base. The interviewees that had experience with SonarQube had the same thoughts, but also explained that the tool is in fact only a tool and should be regarded as such. It should be used in conjunction with other methods of securing code such as code reviews. SonarQube also served to make the code more manageable and the ease to read increased. This would serve to help new people on a project to have an easier time integrating and could help organizations with easier transitions between projects. The graphs shown in the SonarQube interface as shown in Figure 3 SonarQube Dashboard stated to have a positive effect on the motivation to fix code smells and could in turn help with managing the unclean code of the projects. The graph is not only for the motivation says Respondent 5. Its visualization also means that they have an objective improvement according to code smells instead of just "go with gut feeling" as they can see that the graphs have a downward trend. The graphs can also have a positive effect when dealing with product owners and management says Respondent 1. Figure 3 SonarQube Dashboard shows how a dashboard in SonarQube can look. This is something that Respondent 2 talks about, that it is hard to persuade the customer to think about security. "As long as the sun shines, and it might be a survival strategy in us humans, that you don't want to think about other things then." The Respondent elaborates that there are some customers that think about security too much and that can become a problem as well.

"Why would you need redundancy at 4:00 in the night when your employees are sleeping?"

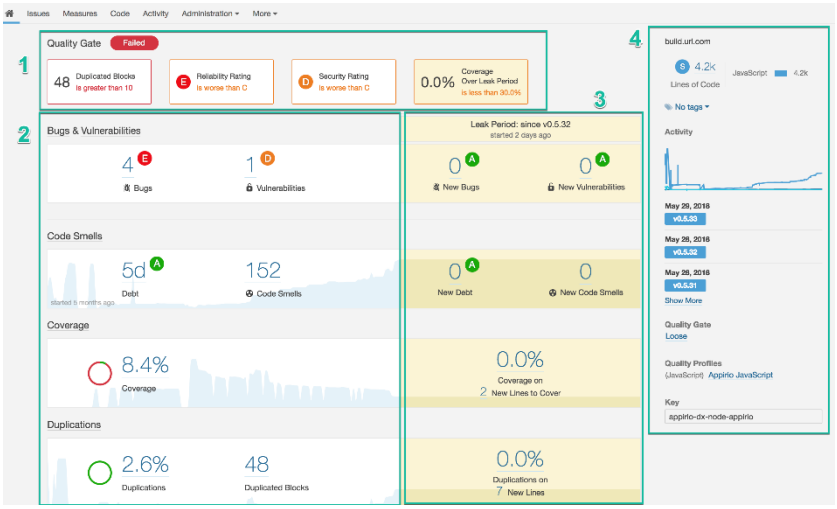

Figure 3 SonarQube Dashboard

Respondent 5 says that thanks to SonarQube, they visit the code on various parts because they need to check what SonarQube accounts for. Which means that, visiting the code itself will improve the code because respondent 5 explains that it is important to leave it in better condition than before when visiting.

Respondent 1 called it, “The Boy Scout rule. Make it a little better while I’m there visiting.”

The integration with continuous improvement is something that all the studied SAST tools were compatible with, and our interviewees explained that they are already using continuous improvement and in turn the integration of SonarQube or similar tools would be a natural inclusion to run when the regular unit and component testing is running, which is being done in all the projects studied from our partner company.

4.2 SAST effects on security

The literature showed that SAST tools were highlighting and, in some cases, suggested fixes to vulnerabilities. There were issues surrounding different SAST tools that found different vulnerabilities. The literature studies and the interviews also strengthen the notion that SAST tools on its own is not an effective way of avoiding vulnerabilities and should be paired with code reviews and other types of security analysis such as Dynamic Application Security Testing.

Respondents 1, 5 and 7 were all adamant that SonarQube was not the most important security tool. And quotes from Respondent 5 and Respondent 1 affirm this. Respondent 1 said “We learn from each other when we do code reviews. It’s almost more important. It’s probably the most important quality work we do.” Respondent 5 to the same affect said “…That we never check in code without another person

looking at it first. It's probably the strongest tool for increasing quality of code."

Respondents 3 and 4 agree that the most important quality work is the code reviews that they have in their specific project. Neither of them uses some sort of security tool but respondent 4 says that it would be a great idea to try it out considering what they have heard about what these types of tools can do.

Respondent 3 also has an open mind to try it and see what the results are, "…It can certainly be useful information". But with that said, the respondent explains that there can also be a possible risk when installing tools like SonarQube because it can generate a huge amount of output that has to be reviewed which will increase the workload a lot, especially when installing it for the first time. The respondent also mentions in the interview, that how can we be certain if SonarQube has better solutions than us according to coding. The respondent says "…Of course it is probably a good approach how to code, but that does not mean that this is the only good way to code. "

SAST, according to both the literature review as well as the interviews, still is a tool that finds security flaws within a system and is an effective tool at finding these bugs, and through constant updates of these tools, more and more vulnerabilities are checked, and errors are shown to the developers who in turn chooses if the error are to be fixed or does not impact the IT system. So, while the tool is excellent at showing what it believes to be wrong, it is still completely up to the developer to do something about it.

When Respondent 2 talked about if SonarQube or similar tools increase security, they said "…The tools on its own does nothing…", and in relation to the tools and the frameworks they implement the respondent said, "…First you need to read the list and understand what it means. And then you need to keep in the back of your mind when you write your own code. And guarantee that you don't [miss them], it doesn't help if you know the pitfalls. So, I believe the tools can play a big role there."

When asked if the effects on security could be measured Respondent 2 said, "…if you decrease the number of vulnerabilities and make sure you have a downward trend, then you have it measured…" The respondent makes sure to be critical about the approach of trusting the security tool while still acknowledging that it logically follows that removing vulnerabilities should increase security.

4.3 Implementing a SAST tool

The implementation of a SAST tool is considered an important part of the lifecycle of the tool. Respondent 2 describes it by saying, "…I think setting up the environment is not trivial for the respective products…" and "…what measurements should we look at, what's an acceptable level…"

They also provide information on if it is a problem of getting the implementation started. When asked if it is hard to convince the customer to consider security they say, “Yes. I think it’s very hard. I also think it’s hard to convince the product owner to think about security.”

The literature provides a picture of some challenges in the implementation and early stages of using a SAST tool. They discuss the continuous integration as something that provides a huge amount of data. A solution provided to this problem was to have frequent smaller checks during daytime and nightly builds that check more thoroughly. A related issue was that vulnerabilities were constantly checked for so if one issue was not resolved before the next run, the error would show up again, and depending on how the environment is set up might cancel the build. They found the solution to this was to omit past errors from future builds. This makes it so that the future builds are not impacted by past faults.

Another issue provided was the education of staff affected by the tool. The developers had a hard time understanding what they had to do when an issue was found. The solution provided to this was to have clear communication and have better future engagement with the team.

4.4 False negatives and positives

False positives are troublesome since time has to be spent troubleshooting code that has no problems whereas false negatives could have disastrous consequences that could crash the program or open it up to malicious actors. As shown in the literature review as well as the interviews, this is a frequent problem in SonarQube and something that must be taken in consideration when using it. Respondent 5 explained that “SonarQube might report things that we decide is actually not wrong.” When asked if overreliance on the tool could occur,

Respondent 1 talked about how complacency might happen when there is over reliance on a tool. This relates to false negatives since only the tool’s capabilities is taken into consideration. They said “I have done all the rules and controls, so now it’s perfect. It’s a risk.”

Respondent 1 also mentions that if SonarQube points out several faults that are not relevant for a specific project there is always a possibility to turn off some rules to avoid being reminded that it is a flaw.

The literature provided in the thesis talks about this problem and one conclusion brought out was that the combination of Static Application Security Testing and Dynamic Application Security Testing should decrease the false negatives since they complement each other.

4.5 Benefits and drawbacks of using a Static Application Security Testing

The benefits are previously described when discussing the security aspects but there are other uses or effects that come from using such a tool other than finding vulnerabilities. The respondents that have previously used a SAST tool talked a lot about manageability of code. They said it's easier for developers to integrate into a project if the guidelines are the same for everyone. It also helps the newly arrived developers to write the same way. It assists in finding code smells which intends to improve the maintainability of the code base.

The drawbacks provided by interviewees are first and foremost surrounding implementation and early stages of using the tool. Respondent 3 described it as, "…always a risk with these types of tools, that you get enormous amounts of output, so a lot of time must be spent going through the results…", they continue with, "…a huge workload that comes from nowhere, particularly when turned on for the first time."

When respondent 1 who has used the tool was asked about the concern they said, "…you get stats on the state of your system…", "…you get numbers on something and can show over time that you are going in the right direction…", and when talking about the "enormous amounts of output" he said, "…it indicates something. Maybe you should change your behaviour or prioritize differently."

Literature provided in the report show that there are both benefits and drawbacks to using a SAST tool. The interviews also strengthen this idea. Below are bullet points showing the negatives and positives of a general SAST tool based on the literature as well as the interviewees opinions.

Benefits:

- Finds bugs and vulnerabilities
- Improves software maintenance
- Helps newer developers write similar code
- Finds code smells
- Helps the developers revisit code and creates a better understanding of the system
- Flexibility in the usage
- Free alternatives
- Free trials to assess if it is worth it
- Might move development down in priority in place of security
- Drives constant improvements

Drawbacks:

- Implementation costs (Economic)

- Over-reliance leads to complacency
- Implementation costs (Time)
- Not effective on its own
- False positives
- False negatives
- A possibility of overwhelming number of errors at once
- Literature found assess that maintenance is not lessened by removing code smells
- Difficult to persuade customers to think about security
- Might move development down in priority in place of security

5 Discussion

This chapter discuss the obtained results from both the literature and interviews. Also, a conclusion is presented. Implications and a method reflection is also discussed.

Both interviews and document studies reflect that developers are curious and willing to try out Static Analysis Security Testing tools. There was hesitation by the interviewees on how it would affect their coding experience. Some sentiments where that SAST tools had the potential to be a nuisance and could be intrusive to the coding experience.

As one respondent mentioned, a fear of installing SAST tools can be the amount of workload it generates which can be a reason why developers never install it. Also, the respondent questioned how to know if SonarQube´s way of coding is better than for example how they code at their project and what fits them.

5.1 Assessing effects on security

The measuring of security is hard, and it is impossible to compare two different IT systems as they are vastly different in every imaginable way. Their use cases, developers, dependencies, and user base are different.

The only way to know if a security tool prevented an attack would be to have gathered data on known attempted attacks and then check if the security tool had previously flagged for that specific vulnerability and in turn the developers patched the vulnerability which led to it not being accessible for exploitation.

The effectivity of a tool could also be evaluated by penetration testing. Penetration testing means ethical hackers who are trained professionally in trying to test security in systems, try to break into the system or use vulnerabilities. If the vulnerabilities could be found and exploited, it points to how the SAST tool found a real, exploitable vulnerability in the system.

Something else that is notable with the SAST tools are that the vulnerabilities presented in the program might have little to no reward for a malicious hacker if they were to break in. A question that could be asked in such cases: Is it worth it to fix vulnerabilities that does not harm the system in case of a break?

The last point is that even though a vulnerability is fixed, there is no way to ever know if that vulnerability would be exploited. Unfortunately, there is no way to tell what would happen in the case that the vulnerability was not fixed. Maybe the system would never be attacked anyway.

Downplaying the vulnerabilities is not the intention, but context matters in these situations and a static analysis tool does not understand the context and flags for everything although some vulnerabilities in some contexts could be a waste of time to fix. This is confirmed by the interviews as well, as the respondents said they have fixes they have decided to ignore.

The respondents who are positive toward SAST tools have responded that they believe that the tools increase security. And this is because it logically follows through removing vulnerabilities that are a risk to the system. Vulnerabilities are classified in OWASP as something that increases the level of danger to a system. So, it could be reasoned that removing the issues that increases the danger to a system should in turn reduce the danger presented, hence increase security. And since OWASP also classifies the severity and frequency of the vulnerability, it should help with prioritization of the vulnerabilities and in turn help the most important vulnerabilities to be patched first.

Through the interviews we can see a common thread, and that is the importance of having in mind that the tool is just there to help and not rely on it too much. And this is important to keep in mind as implementation of these tools are done. They are just, as Respondent 1 put it, "…dumb tools."

A good idea that was mentioned about these types of tools, that appeared through the interviews, is that it will help developers follow the same code pattern, which will result in an easy way of being introduced to the code base for new employees. This should help with increasing the efficiency when introducing employees, as well as help developers that are experienced with the system to look at older code and understand it, since it follows the same structure.

A benefit of SonarQube is that it can visualize improvements using graphs which will make it easier to communicate with different stake holders. This is something that is very important, since communication with stake holders are what keeps the organization going. This visualization and communication tool can also help with introducing the tool to another project and product owner. A graph can be shown from a project that has previously used the SAST tool and show that their numbers of vulnerabilities and code smells has gone down by 50% in a year, for example. This might help persuade the product owner to take into consideration a SAST tool. Respondents have also expressed that the graph helps with motivation in fixing the issues, and it helps to remind them that there are issues.

5.2 Implications

Although there are no direct societal consequences of our research, it shines a light on security tools. This might have indirect effect on some companies, mainly our partner company, Triona. The research could help with showing as well as solving some of the issues in im-

plementation of these tools and could do the same to other companies which use the research as an aid when looking at security implementations. In an ideal world this might in turn help companies and society be more conscious about security as well as the SAST tools.

There should be no ethical consequences to our research. Our research has no details of confidential information or examples of open vulnerabilities that could cause harm. Arguments could be made that misuse of tools we have talked positively about could lead to increased costs as well as overreliance on the tools. This has been explained in the paper to be something to be taken into consideration when using the tools.

Another ethical implication of our research could be the expectation of continued work in the company. As both authors have been offered employment as well as financial compensation after the thesis is done, it could be argued that the authors receive financial benefits from the work.

5.3 Method reflection

According to (Oates, 2005) document-based research is a useful source of information which is easy to obtain. It also allows to look at many sources within the same subject and to evaluate them based on the surrounding circumstances. These circumstances are also what makes document-based research volatile. Important factors are who and how the document was produced and what the purpose of the document was. (Oates, 2005) also talks about the notion that document-based research is seen as "…official, authoritative-looking…" and this sentiment often makes is seem like the documents can give an objective picture of reality, which is not the case.

Our main sources that try and describe reality with data, are peer-reviewed and the data collected is primarily dependable. We use sources that come from companies that describe their own products or artifacts, where an elevated level of scepticism should be used, since their goal often is to sell idea of their product.

Something that was realized late in the research and that could have given a clearer picture of the results was that we only had qualitative gathered data through interviews. The approach could have been improved by doing qualitative interviews and combined them with quantitative questionaries.

6 Conclusion

In this section we conclude our thesis with the conclusion of our study by answering the research questions.

6.1 Do Static Application Security Testing (SAST) tools help in writing secure code?

SAST tools are an effective way to find vulnerabilities in code. It should not be used solely and works best in conjunction with other security measures such as code reviews and dynamic application security testing.

6.2 What are the benefits and drawbacks of using a SAST tool?

The benefits and drawbacks of using Static Application Security Testing are described below.

Benefits:

- Finds bugs and vulnerabilities
- Improves software maintenance
- Helps newer developers write similar code
- Finds code smells
- Helps the developers revisit code and creates a better understanding of the system
- Flexibility in the usage
- Free alternatives
- Free trials to assess if it is worth it
- Might move development down in priority in place of security
- Drives constant improvements

Drawbacks:

- Implementation costs (Economic)
- Over-reliance leads to complacency
- Implementation costs (Time)
- Not effective on its own
- False positives
- False negatives
- A possibility of overwhelming number of errors at once
- Literature found assess that maintenance is not lessened by removing code smells
- Difficult to persuade customers to think about security
- Might move development down in priority in place of security

7 Further research

This chapter discuss further research that can be interesting.

There are other studies that have compared SAST tools, but they are somewhat old, and since technology moves quickly the tools explored are now largely obsolete. A mapping of the current tools and comparing them through test cases could be interesting. This could be done through using the frameworks explained in this thesis and use the top 25, or top 10 and then testing all the tools. Then a measurement could be done that would compare the prices, efficiency, and speed of the tools.

A subject that was brought up in two interviews are the prominence of cloud vs local. And this subject is big but could be focused down into looking at the different security requirements and vulnerabilities of the two, as well as how the implementations could differ.

Another idea that was found through the paper were Dynamic Application Security Testing. And this was often brought up as a complement to SAST. But there are different implementations within this category, and it could be interesting to look at them all, as well as see if they are efficient or not.

A case study could work as an approach as well, where looking at a company that previously had security issues could lead to discovery of how SAST or DAST tools could have prevented or at least lessened the impact of an IT attack or mishap.

The idea of cost is something that this report does not account for, and this was an initial idea considered in the report. The authors had the thought that it could be checked for later in the research, and realized that to get price for the products, contact with the companies was required, since few companies had their prices written out on their pages. This is also something that must be compared according to lines of code as well as the size of the teams on projects, and the size of the company. Different tools have different criteria for cost so this also makes it an interesting and somewhat challenging notion.

8 References

1, R. (2022, April 12). Security tools. (E. Seger, & F. Schedin, Interviewers)

Baset, A. Z., & Denning, T. (2017). IDE Plugins for Detecting Input-Validation Vulnerabilities. *2017 IEEE Symposium on Security and Privacy Workshops* (pp. 143-146). Utah: IEEE.

Bolduc, C. (2016). Lessons Learned: Using a Static Analysis Tool Within a Continuous Integration System. *IEEE 27th International Symposium on Software Reliability Engineering Workshops*, 37-40.

Common Weakness Enumeration. (2022, April 4). *CWE-125: Out-of-bounds Read*. Retrieved from Common Weakness Enumeration: https://cwe.mitre.org/data/definitions/125.html

Confessore, N. (2018, April 4). Cambridge Analytica and Facebook: The Scandal and the Fallout So Far. *New York Times*.

JetBrains. (2022, April 20). *ReSharper features*. Retrieved from ReSharper: https://www.jetbrains.com/resharper/features/

Johnson, S. C. (1978). *Lint, a C Program Checker.* New Jersey: Bell Laboratories.

Kiuwan. (2020, March 11). *Getting started with Kiuwan*. Retrieved from Kiuwan: https://www.kiuwan.com/docs/display/K5/Getting+Started+with+Kiuwan#GettingStartedwithKiuwan-HowKiuwanworks

Lebanidze, E. (2008). *The Need for Fourth Generation Static Analysis Tools for Security - From Bugs to Flaws.* Ghent: OWASP.

Lenarduzzi, V., Lomio, F., Huttunen, H., & Taibi, D. (2020). *Are SonarQube Rules Inducing Bugs?* London: SANER.

Li, P., & Cui, B. (2010). A comparative study on software vulnerability static analysis techniques and tools. *IEEE International Conference on Information Theory and Information Security*, 521-524.

Mahalingam, M. (2014). *Learning Continuous Integration with TeamCity.* Birmingham: Packt Publishing Ltd.

Marcilio, D., Bonifacio, R., Monteiro, E., Canedo, E., Luz, W., & Pinto, G. (2019). Are Static Analysis Violations Really Fixed? A Closer Look at Realistic Usage of SonarQube. *International Conference on Prgram Comprehension* (pp. 209-219). IEEE.

Microsoft. (2021, May 14). *What is continuous integration?* Retrieved from Azure DevOps: https://docs.microsoft.com/en-us/devops/develop/what-is-continuous-integration

Microsoft. (2021, July 7). *What is DevOps?* Retrieved from Azure DevOps: https://docs.microsoft.com/en-us/devops/what-is-devops

Microsoft. (2022, April 20). *Intellisense in Visual Studio*. Retrieved from Microsoft: https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2022

Microsoft. (2022, May 10). *What is .Net?* Retrieved from Microsoft: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

National Institute of Standards and Technology. (2022, May 10). *National Institute of Standards and Technology*. Retrieved from National Institute of Standards and Technology: https://www.nist.gov/

National Institute of Standards and Technology. (2022, April 01). *National vulnerability database* . Retrieved from National Institute of Standards and Technology: https://nvd.nist.gov/

National institute of standards and technology. (2022, April 01). *National vulnerability database, CVCC*. Retrieved from National institute of standards and technology: https://nvd.nist.gov/vuln-metrics/cvss

National Instruments Corp. (2019, 5 March). *What is Manageability?* Retrieved from National Instruments: https://www.ni.com/sv-se/innovations/white-papers/13/what-is-manageability-.html#section-529388437

Oates, B. J. (2005). *Researching Information Systems and Computing.* SAGE Publications Ltd.

OWASP. (2022, April 4). *A01 Broken Access Control*. Retrieved from OWASP: https://owasp.org/Top10/A01_2021-Broken_Access_Control/

OWASP. (2022, April 4). *A03 Injection*. Retrieved from OWASP: https://owasp.org/Top10/A03_2021-Injection/

OWASP. (2022, April 4). *Cryptographic Failures*. Retrieved from OWASP: https://owasp.org/Top10/A03_2021-Injection/

OWASP. (2022, March 29). *The Making of the OWASP Top Ten*. Retrieved from OWASP Top Ten: https://www.owasptopten.org/themakingoftheowasptopten

Oyetoyan, T., Milosheska, B., Grini, M., & Soares, C. D. (2018). Myths and Facts About Static Application Security Testing Tools: An Action Research at Telenor Digital. *Internation Conference on Agile Software Development* (pp. 86-103). Springer.

Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., & De Lucia, A. (2018). [Journal First] On the Diffuseness and the Impact on Maintainability of Code Smells: A Large Scale Empirical Investigation. *IEEE/ACM 40th International Conference on Software Engineering* , 482-482.

PARASOFT. (2019, May 15). *About Parasoft dotTest*. Retrieved from Parasoft dotTEST: https://docs.parasoft.com/display/DOTTEST1041/Parasoft+DTP+Engines

PVS-Studio. (2022, May 25). *PVS-Studio*. Retrieved from PVS-Studio analyzer: https://pvs-studio.com/en/pvs-studio/

Red Hat. (2019, January 8). *What is and IDE?* Retrieved from Red Hat: https://www.redhat.com/en/topics/middleware/what-is-ide

Rooney, P. (2002, October 3). Microsoft's CEO: 80-20 Rule Applies To Bugs, Not Just Features. *CRN*.

Security Reviewer. (2022, March 28). *Static Reviewer*. Retrieved from Security Reviewer: https://securityreviewer.atlassian.net/wiki/spaces/KC/pages/196633/Static+Reviewer

Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access 5*, 3909-3943.

Sjøberg, D. I., Yamashita, A., Anda, B. C., Mockus, A., & Dybå, T. (2013). Quantifying the Effect of Code Smells on Maintenance Effort. *IEEE Transactions on Software Engineering*, 1144-1156.

SonarLint. (2022, April 20). *Features*. Retrieved from SonarLint: https://www.sonarlint.org/features

SonarQube. (2022, April 1). *SonarQube Documentation*. Retrieved from SonarQube: https://docs.sonarqube.org/latest/

The MITRE corporation. (2021, December 23). *About CWE*. Retrieved from Common Weekness Enumeration: https://cwe.mitre.org/about/index.html

The MITRE Corporation. (2021, July 20). *CWE-787: Out-of-bounds Write*. Retrieved from Common Weakness Enumeration: https://cwe.mitre.org/data/definitions/787.html

The MITRE Corporation. (2021, October 26). *CWE-79*. Retrieved from Common Weakness Enumeration: https://cwe.mitre.org/data/definitions/79.html

The MITRE corporation. (2022, April 01). *CVE Program mission*. Retrieved from Common vulnerabilities and exposure: https://www.cve.org/

Triona. (2022, May 23). *TRACS Flow*. Retrieved from Triona: https://www.triona.se/produkter_tjanster/produkter/tracs_flow/

Triona. (2022, May 23). *Transport Network Engine*. Retrieved from Triona: https://www.triona.se/produkter_tjanster/produkter/transport_network_engine/

van Emden, E., & Moonen, L. (2002). Java quality assurance by detecting code smells,. *Ninth Working Conference on Reverse Engineering, 2002*, 97-106.

Yang, J., Tan, L., & Kristofer A Duer, J. P. (2019). Toward Better Utilizing Static Application Security Testing. *Internation Conference on Software Engineering: Software Engineering in Practice* (pp. 51-60). IEEE.

Appendix A: Search

Author	Title	Year	Keyword/s	Type of source	Found by
Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., & De Lucia, A.	On the Diffuseness and the Impact on Maintainability of Code Smells: A Large Scale Empirical Investigation.	2018	Code smells, maintenance	Journal article	Google Scholar
van Emden, E., & Moonen, L.	Java quality assurance by detecting code smells	2002	Code smells, code smells maintenace	Conference article	Google scholar
Yang, J., Tan, L., & Kristofer A Duer, J. P.	Toward Better Utilizing Static Application Security Testing.	2019	Static Application Security Testing, effects, challenges	Conference article	Google scholar
Shahin, M., Babar, M. A., & Zhu, L.	Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices.	2017	Continuous integration, challenges, implemenation	Journal Article	Google scholar
Rangnau, T., Buijtenen, R. v., Fransen, F., & Turkmen, F.	Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines.	2020	Dynamic Application Security Testing, Implementation	Conference article	Google scholar
Baset, A. Z., & Denning, T.	IDE Plugins for Detecting Input-Validation Vulnerabilities.	2017	IDE Static Analysis	Conference article	Google Scholar
Oates, B. J.	Researching Information Systems and Computing.	2005	Researching information systems	Book	Used throughout the program, found on Summon

Appendix B: Interview questions

Respondent 1:

Vad har du för branscherfarenheter inom systemutveckling/IT?

Har du någon erfarenhet med att använda något säkerhetsverktyg från tidigare projekt/arbeten?

Vad är Graphical Development Tool? Inom vilket område finns detta projekt?

Vad är din arbetsroll?

Hur länge har du jobbat med Graphical Development Tool? Hur länge har du haft den roll du har nu?

Finns det något du tycker är extra roligt inom projektet?

Vad använder ni för plattform inom projektet?

Arbetar du något med säkerheten inom projektet?

Vad betyder säkerhet inom IT för dig?

Använder ni något verktyg för säkerhet inom projektet?

Hur kommer det sig att ni implementerade det säkerhetsverktyget på Graphical Development Tool?

Vad tycker du fungerar bra med verktyget ni använder i dag?

Vad tycker du kan förbättras med verktyget ni använder idag?

Vad är skillnaden på hur ni använder verktyget på detta projekt från de föregående projekt du jobbat på?

Hur arbetar ni med verktyget? Är det något som är utsedd att hålla kolla på hur verktyget fungerar? Och följer upp brister om de uppstår.

Vad är skillnaderna från när ni inte använde verktyget?

Är det någon skillnad på hur ni använder verktyget från när ni började använda verktyget från idag?

Känner du till Intellisense, Sonarlint eller Resharper som är automatiska kodkompletteringsverktyg?

Vad tycker du om dessa?

Hur tror du framtiden kommer se ut för säkerhetsverktyg? Eller säkerhet generellt inom IT?

Har du något att tillägga som vi inte tagit upp?

Har du någon person som du tror skulle vara intressant att prata mer med om detta? Fråga till den personen?

Om du kommer på något som kan vara av intresse för oss som inte har tagits upp, eller om du märker att du har missat något är det bara att maila en av oss.

Respondent 2:

Vad har du för branscherfarenheter inom systemutveckling/IT?

Har du någon erfarenhet med att använda något säkerhetsverktyg från tidigare projekt/arbeten?

Vad tycker du om automatiska kodkompletteringsverktyg som till exempel Intellisense, Sonarlint och Resharper?

Vad gör du inom din roll? Inom vilka områden arbetar du?

Finns det något du tycker är extra roligt med att vara CTO?

Vad betyder säkerhet inom IT för dig?

Hur jobbar ni på Triona kring säkerhetsverktyg? Får varje projekt inom Triona bestämma själva hur dom gör eller har ni några bestämda regler kring hur sådana verktyg ska användas?

Prioriterar man att använda ett säkerhetsverktyg för flera projekt för att det ska vara lätt för utvecen att byta mellan projekten? Eller skräddarsyr man lösningen mer efter varje enskilt projekt?

Vad tycker du skiljer de olika projekten i termer av säkerhet? (Har vissa projekt en högre standard?) Varför?

Vad tycker du om Resharper, det automatiska kodkompletteringsprogrammet ni använder idag?

Hur upplever du vad utvecklarna på projektet tycker om Resharper?

Anser du att det är värt kostnaderna?

Var du med när Resharper införskaffades?

Vad var anledningen till att ni valde Resharper?

Hur utvärderar ni vilka verktyg som behövs för vilka projekt? Har ni någon uppföljning för de verktyg som används idag?

Hur avgör man om ett verktyg fungerar eller inte? Om man till exempel har ett år där man testar ett verktyg.

Får du reda på om verktygen har hittat säkerhetsbrister inom de olika projekten? I så fall hur utvärderas dom?

Anser du att dessa verktyg, som till exempel Resharper och SonarQube, gör att man får säkrare system?

Har ni funderat på att implementera några säkerhetsverktyg inom Tracs Flow och TNE?

Varför, varför inte?

Vilka för och nackdelar finns vid implementation av dessa säkerhetsverktyg?

Är det värt jobbet i sig att installera säkerhetsverktyg så som SonarQube? Börja beta av klumpen

Tycker du man borde ha en avsatt tid för säkerhet inom projekten? Är det realistiskt?

Hur tror du framtiden kommer se ut för säkerhetsverktyg? Eller säkerhet generellt inom IT?

Har du något att tillägga som vi inte tagit upp?

Har du någon person som du tror skulle vara intressant att prata mer med om detta? Fråga till den personen?

Om du kommer på något som kan vara av intresse för oss som inte har tagits upp, eller om du märker att du har missat något är det bara att maila en av oss.

Respondent 3:

Vad har du för branscherfarenheter inom systemutveckling/IT?

Har du någon erfarenhet med att använda något säkerhetsverktyg från tidigare projekt/arbeten?

Vad är TNE? Inom vilket område finns detta projekt?

Hur länge har du jobbat med projektet?

Vad har du för arbetsroll? Hur länge har du haft den rollen?

Finns det något du tycker är extra roligt inom projektet?

Vad använder ni för plattform för projektet?

Vad betyder säkerhet inom IT för dig?

Arbetar ni med något säkerhetsverktyg idag?

Känner du till något säkerhetsverktyg? Hur? Vad har du hört?

Hur tänker ni kring säkerhet när ni inte använder ett verktyg?

Tycker du att ni har ett behov av ett säkerhetsverktyg?

Hur vill du att säkerhetsverktyget ska fungera?

Känner du till några säkerhetsbrister idag?

Vi vet att du är duktig på TeamCity och Continuous Integration, har du kollat på någon säkerhetsimplementation inom detta?

Känner du till Intellisense, Sonarlint eller Resharper som är automatiska kodkompletteringsverktyg?

Vad tycker du om dessa?

Hur tror du framtiden kommer se ut för säkerhetsverktyg? Eller säkerhet generellt inom IT?

Har du något att tillägga som vi inte tagit upp?

Har du någon person som du tror skulle vara intressant att prata mer med om detta? Fråga till den personen?

Om du kommer på något som kan vara av intresse för oss som inte har tagits upp, eller om du märker att du har missat något är det bara att maila en av oss.

Respondent 4:

Vad har du för branscherfarenheter inom systemutveckling/IT?

Har du någon erfarenhet med att använda något säkerhetsverktyg från tidigare projekt/arbeten?

Vad är Tracs Flow? Inom vilket område finns detta projekt?

Hur länge har du jobbat med projektet?

Vad har du för arbetsroll? Hur länge har du haft den rollen?

Finns det något du tycker är extra roligt inom projektet?

Vad använder ni för plattform för projektet?

Vad betyder säkerhet inom IT för dig?

Arbetar ni med något säkerhetsverktyg idag?

Känner du till något säkerhetsverktyg? Hur? Vad har du hört?

Hur tänker ni kring säkerhet när ni inte använder ett verktyg?

Känner du till några säkerhetsbrister idag?

Tycker du att ni har ett behov av ett säkerhetsverktyg?

Hur vill du att säkerhetsverktyget ska fungera?

Känner du till Intellisense, Sonarlint eller Resharper som är automatiska kodkompletteringsverktyg?

Vad tycker du om dessa?

Hur tror du framtiden kommer se ut för säkerhetsverktyg? Eller säkerhet generellt inom IT?

Har du något att tillägga som vi inte tagit upp?

Har du någon person som du tror skulle vara intressant att prata mer med om detta? Fråga till den personen?

Om du kommer på något som kan vara av intresse för oss som inte har tagits upp, eller om du märker att du har missat något är det bara att maila en av oss.

Respondent 6:

Vad har du för branscherfarenheter inom systemutveckling/IT?

Har du någon erfarenhet med att använda något säkerhetsverktyg från tidigare projekt/arbeten?

Vad är Tracs Flow? Inom vilket område finns detta projekt?

Hur länge har du jobbat med projektet?

Vad har du för arbetsroll? Hur länge har du haft den rollen?

Finns det något du tycker är extra roligt inom projektet?

Vad använder ni för plattform för projektet?

Vad betyder säkerhet inom IT för dig?

Arbetar ni med något säkerhetsverktyg idag?

Känner du till något säkerhetsverktyg? Hur? Vad har du hört?

Hur tänker ni kring säkerhet när ni inte använder ett verktyg?

Tycker du att ni har ett behov av ett säkerhetsverktyg?

Hur vill du att säkerhetsverktyget ska fungera?

Känner du till några säkerhetsbrister idag?

Tror du något säkerhetsverktyg hade kunnat hitta bristen innan det kom ut till kund?

Känner du till Intellisense, Sonarlint eller Resharper som är automatiska kodkompletteringsverktyg?

Vad tycker du om dessa?

Hur tror du framtiden kommer se ut för säkerhetsverktyg? Eller säkerhet generellt inom IT?

Har du något att tillägga som vi inte tagit upp?

Har du någon person som du tror skulle vara intressant att prata mer med om detta? Fråga till den personen?

Om du kommer på något som kan vara av intresse för oss som inte har tagits upp, eller om du märker att du har missat något är det bara att maila en av oss.

Respondent 5 & 7:

Vad har du för branscherfarenheter inom systemutveckling/IT?

Har du någon erfarenhet med att använda något säkerhetsverktyg från tidigare projekt/arbeten?

Vad är Graphical Development Tool? Inom vilket område finns detta projekt?

Hur länge har du jobbat med projektet?

Vad har du för arbetsroll? Hur länge har du haft den rollen?

Vad använder ni för plattform för projektet?

Finns det något du tycker är extra roligt inom projektet?

Vad betyder säkerhet inom IT för dig?

Arbetar ni med något säkerhetsverktyg?

Hur tycker du det fungerar?

Hur arbetar ni med verktyget? Är det något som är utsedd att hålla kolla på hur verktyget fungerar? Och följer upp brister om de uppstår.

Finns det några nackdelar att arbeta med SonarQube?

Vad är skillnaderna från när ni inte använde verktyget?

Är det någon skillnad på hur ni använder verktyget från när ni började använda verktyget från idag?

Känner du till Intellisense, Sonarlint eller Resharper som är automatiska kodkompletteringsverktyg?

Vad tycker du om dessa?

Hur tror du framtiden kommer se ut för säkerhetsverktyg? Eller säkerhet generellt inom IT?

Har du något att tillägga som vi inte tagit upp?

Har du någon person som du tror skulle vara intressant att prata mer med om detta? Fråga till den personen?

Om du kommer på något som kan vara av intresse för oss som inte har tagits upp, eller om du märker att du har missat något är det bara att maila en av oss.

Appendix C: Static Application Security Testing Tools

Name of tool	TeamCity	Azure Devops	CWE Top 10	OWASP Top 25	Open Source	.Net Compatible	Free alternative	Requires other implementations(Such as Bamboo)	During build	Autofix
Static Reviewer	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No
Kiuwan	Yes	Yes	Yes	Yes	No	Yes	No	No	Yes	No
dottest	Yes	No	Yes	Yes	No	Yes	No	No	Yes	No
SonarQube	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No
PVS-Studio	Yes	Yes	Yes	Yes	No	Yes	No	No	Yes	No
Deepsource	No	No	Yes	Yes	No	No	No	No	No	Yes