*Article*

# Security Monitoring during Software Development: An Industrial Case Study

**Miltiadis Siavvas [1,*], Dimitrios Tsoukalas [1], Ilias Kalouptsoglou [1], Evdoxia Manganopoulou [2], Georgios Manolis [2], Dionysios Kehagias [1] and Dimitrios Tzovaras [1]**

[1] Centre for Research and Technology Hellas, 57001 Thessaloniki, Greece; tsoukj@iti.gr (D.T.); iliaskaloup@iti.gr (I.K.); diok@iti.gr (D.K.); dimitrios.tzovaras@iti.gr (D.T.)
[2] Onelity Hellas MEPE, 57001 Thessaloniki, Greece; evdoxia.manganopoulou@onelity.com (E.M.); georgios.manolis@onelity.com (G.M.)
* Correspondence: siavvasm@iti.gr

**Abstract:** The devastating consequences of successful security breaches that have been observed recently have forced more and more software development enterprises to shift their focus towards building software products that are highly secure (i.e., vulnerability-free) from the ground up. In order to produce secure software applications, appropriate mechanisms are required for enabling project managers and developers to monitor the security level of their products during their development and identify and eliminate vulnerabilities prior to their release. A large number of such mechanisms have been proposed in the literature over the years, but limited attempts with respect to their industrial applicability, relevance, and practicality can be found. To this end, in the present paper, we demonstrate an integrated security platform, the VM4SEC platform, which exhibits cutting-edge solutions for software security monitoring and optimization, based on static and textual source code analysis. The platform was built in a way to satisfy the actual security needs of a real software development company. For this purpose, an industrial case study was conducted in order to identify the current security state of the company and its security needs in order for the employed security mechanisms to be adapted to the specific needs of the company. Based on this analysis, the overall architecture of the platform and the parameters of the selected models and mechanisms were properly defined and demonstrated in the present paper. The purpose of this paper is to showcase how cutting-edge security monitoring and optimization mechanisms can be adapted to the needs of a dedicated company and to be used as a blueprint for constructing similar security monitoring platforms and pipelines.

**Keywords:** software security; security by design; security monitoring; verification and validation; vulnerability prediction; experience report

check for updates

**Citation:** Siavvas, M.; Tsoukalas, D.; Kalouptsoglou, I.; Manganopoulou, E.; Manolis, G.; Kehagias, D.; Tzovaras, D. Security Monitoring during Software Development: An Industrial Case Study. *Appl. Sci.* **2023**, *13*, 6872. https://doi.org/10.3390/app13126872

Academic Editor: Paolino Di Felice

Received: 28 April 2023
Revised: 28 May 2023
Accepted: 2 June 2023
Published: 6 June 2023

## 1. Introduction

In the traditional software development lifecycle (SDLC), security is treated as an afterthought. In particular, software products are built without having security in mind, and is added during their deployment phase through the addition of external protection mechanisms, such as attack detection, firewalls, etc., in an attempt to prevent malicious individuals from exploiting existing vulnerabilities [1,2]. Although this approach was considered sufficient for common offline software applications, the observed transition to the cloud and the software-as-a-service (SaaS) paradigm that makes the software applications accessible via the Internet, along with the fact that external protection mechanisms can be bypassed regardless of their strength, showcase the need for a more careful consideration of security aspects during the overall SDLC. The above reasons, along with the numerous real-world examples of software vulnerabilities that led to devastating financial and reputation damages (e.g., Equifax Breach [3], HeartBleed [4], WannaCry [5], Log4j [6], etc.), forced software companies to shift their focus towards the security-by-design paradigm, that is, on adopting a secure SDLC that leads to the construction of software applications that are highly secure (i.e., vulnerability-free) from their ground up.

The vast majority of software vulnerabilities are introduced during the coding phase of the SDLC through coding mistakes that are made by the developers [2,7,8], either due to their lack of security expertise or due to strict production deadlines that force them to make compromises on the quality of their produced code [9,10]. Hence, appropriate tools are required to help developers avoid the introduction of such issues during the development and project managers better monitor the security of developed source code. Static analysis is considered one of the most effective mechanisms for detecting potential vulnerabilities during the overall development, and therefore it is a must-have feature for all of the popular Secure SDLCs, including Microsoft's SDL [11], Cigital's Touchpoints [1], and OWASP OpenSAAM (http://www.opensamm.org/). However, the main problem of static analysis tools is that they produce long lists of raw warnings that, while encapsulating important security information for the analyzed software, are difficult to comprehend by technical and non-technical stakeholders. Therefore, there is a need for knowledge extraction tools that are able to post-process these results in order to extract useful security-relevant information and present it in a more attractive and easy-to-understand way.

Several mechanisms have been proposed in the literature over the years for enabling the security monitoring of software products during their development, by leveraging security-related results produced by static analysis. Two such mechanisms that have attracted the attention of the research community are (i) the security assessment models, which are models that provide high-level metrics that reflect important security aspects of the analyzed software that are computed by aggregating the results of static analysis, and (ii) the vulnerability prediction models, which are models that are able to detect security hotspots, i.e., software components that are likely to contain vulnerabilities. Both models can be leveraged by project managers for facilitating their decision-making activities during the SDLC. In fact, they allow them to better monitor the security of the developed software and identify those security aspects or components that require immediate attention by the development team. However, the main shortcoming of these mechanisms, which also explains their limited adoption in practice, is that they are difficult to set up and to be tailored to the needs of a specific software company. There is a need for a close collaboration between security experts/researchers and the interested software companies in order to properly configure such security monitoring models, so that the companies can leverage their benefits. There is also a need for defining an approach that can be followed for properly adapting these novelties to the pipelines of software development companies.

To this end, in the present paper, we conduct an industrial case study that demonstrates how novel security monitoring mechanisms can be turned into a unified practical solution properly configured to satisfy the needs of a specific software development company, in order to be used in practice for monitoring the security of its developed software products, towards its transition to a more secure SDLC. In particular, in the present paper, we demonstrate an integrated security monitoring platform, the *VM4SEC platform*, which exhibits cutting-edge security assessment and vulnerability prediction mechanisms that have been developed in order to meet the needs of a software development house. As a first step towards building the platform, a case study was conducted based on a survey and a focus group with key employees of the company, in order to identify the current security state of the company, its security needs, the most suitable security monitoring mechanisms that need to be deployed, and the desired functionalities. Subsequently, based on the retrieved feedback, the functional and non-functional requirements of the platform were extracted, as well as the main parameters of the mechanisms were defined in order to be tailored to the company needs. Finally, the final architecture and deployment diagrams were defined, and the proposed solutions were deployed and started being used by the company.

The present paper, apart from showcasing the VM4SEC security monitoring platform, can be also used as an example on how similar novel security monitoring solutions can be made operational in the form of practical tools that are properly configured, in order to satisfy the needs of a given company. This is considered important, since the lack of customizability, configurability, and adaptability are some of the main reasons that prevent software companies from utilizing security monitoring mechanisms in practice [12,13], despite the acknowledged benefits that they can provide in improving the security of their produced software.

At this point, it should be noted that although several static analysis platforms exist on the market that are able to detect security issues, none of them provide high-level quantitative security measures that reflect important internal security aspects of the analyzed software, nor its overall security level, even though novel security assessment models have been proposed lately [14–18] . In addition to this, no operationalized vulnerability prediction model (VPM) exists in the literature that can be used directly in practice for identifying the existence of potentially vulnerable components in software products under development, despite the recent advancements in the field of vulnerability prediction [19–29]. To the best of our knowledge, the VM4SEC platform is the only platform that provides novel quantitative security assessment and vulnerability prediction models that can be used directly in practice for monitoring and optimizing the security of software products. Most importantly, this is the first research attempt that presents a formal approach for adapting and tailoring such novel security monitoring solutions to fit the needs of a specific company or application domain. This is considered important since such novel security monitoring solutions require proper configuration and adaptation before being used in practice by actual companies, which is a tedious process that often distracts companies from using such mechanisms in practice.

To facilitate the readability and the understandability of the present paper, a road map of the overall methodology that was employed for building the VM4SEC platform is illustrated in Figure 1. As can be seen by Figure 1, the overall methodology consists of five sequential steps. Initially, a two-step survey and a focus group were conducted in order to understand the current security state of the company and its security needs. Subsequently, the requirements of the platform were elicited from the survey and formally expressed using tables and diagrams. Then, the cutting-edge security monitoring mechanisms were built and configured properly in order to satisfy the needs of the company, as expressed in their requirements. Finally, a plan for the implementation and deployment of the VM4SEC platform was devised. A detailed description of these five steps is provided in Section 3.

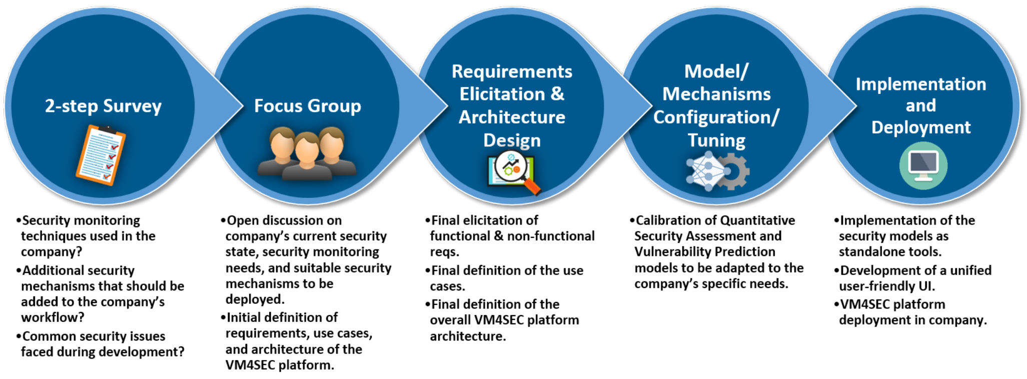

2-step Survey
•Security monitoring techniques used in the company?
•Additional security mechanisms that should be added to the company's workflow?
•Common security issues faced during development?
Focus Group
•Open discussion on company's current security state, security monitoring needs, and suitable security mechanisms to be deployed.
•Initial definition of requirements, use cases, and architecture of the VM4SEC platform.
Requirements Elicitation & Architecture Design
•Final elicitation of functional & non-functional reqs.
•Final definition of the use cases.
•Final definition of the overall VM4SEC platform architecture.
Model/ Mechanisms Configuration/ Tuning
•Calibration of Quantitative Security Assessment and Vulnerability Prediction models to be adapted to the company's specific needs.
Implementation and Deployment
•Implementation of the security models as standalone tools.
•Development of a unified user-friendly UI.
•VM4SEC platform deployment in company.

**Figure 1.** The high-level overview of the overall methodology.

The rest of the paper is structured as follows: Section 2 discusses the related work in the field. Section 3 discusses in detail the methodology that was followed for the present study. In particular, it gives an overview of (i) the survey and the focus group that have been conducted for gathering information from the company, (ii) the process that was followed for eliciting the requirements of the VM4SEC platform and designing its overall architecture,

(iii) the core security monitoring mechanisms that were deployed and how their parameters were computed, and (iv) the final implementation and deployment diagram of the VM4SEC platform that was derived through this process. Finally, Section 4 concludes the paper and discusses directions for future work.

## 2. Related Work

White box testing techniques, and particularly static code analysis, have been found effective in identifying security issues (i.e., vulnerabilities) that reside in the source code of software products [30–32]. Several individual static analysis tools and broader static analysis platforms have been proposed over the years, which are able to detect specific types of security issues among other bugs (e.g., [33]). Static analysis tools process the source code of the analyzed software in an attempt to find predefined patterns that indicate the existence of vulnerabilities. In order to identify potential security weaknesses, they apply a wide range of analysis techniques ranging from simple text processing and pattern matching with the use of regular expressions, to more advanced graph-based techniques such as Abstract Syntax Trees (ASTs), Data-Flow Graphs (DFGs), Control-Flow Graphs (CFGs), and Code-Property Graphs (CPGs), in which they traverse the graphs searching for violations of specific rules or the existence of known vulnerability patterns. The output of these tools is a list of warnings (i.e., alerts) that indicate a potential security issue, providing relevant information about the issue, including its type, severity, and location in the source code.

A known shortcoming of existing static code analyzers, which is known to hinder their wider adoption in practice, is the generation of long lists of raw warnings [34–36]. These warnings, although they contain important information for the security of the analyzed software, are in a raw form that is highly difficult to be comprehended by non-technical experts such as project managers [31,34]. Hence, appropriate techniques are required on top of the results of static analysis, in order to facilitate decision-making during software development [13,37]. Particularly, the results of static analysis could be leveraged for conducting quantitative security evaluation, providing high-level security metrics, which are easier to understand than the low-level warnings and can reflect important security aspects of the analyzed software. Several models and techniques have been proposed in the past years for providing quantitative security evaluation of software products based on the results of static analysis, demonstrating promising results [14–18]. However, none of the existing and widely used static analysis tools and platforms provide such quantitative evaluation models and techniques. They use only visualization techniques in order to provide more intuitive summaries of the identified security issues, but they do not provide high-level security metrics that reflect important aspects of the security of the analyzed software (e.g., Confidentiality, Integrity, Availability, etc.).

Apart from quantitative security assessment, the results of the white box testing of a software product's source code can be leveraged for providing other advanced security monitoring mechanisms. One such mechanism that has recently attracted the attention of the research community is vulnerability prediction. Vulnerability prediction focuses on predicting the existence of vulnerabilities in software applications, and in fact, software components. It is based on the construction of vulnerability prediction models (VPMs), which are machine learning (ML) models that are able to predict whether a given software component may contain vulnerabilities or not [19,38,39]. Several models have been proposed over the years, utilizing various software-related attributes as input features (i.e., predictors), including software metrics [19–21] and text features [22–29]. Among them, the text-mining-based VPMs have demonstrated the most promising results (i.e., highest predictive performance). Initially, text-mining-based VPMs were based on simple ML models text mining techniques such as Bag of Words (BoW) [22], whereas recently, more advanced techniques have been utilized, including Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) models that use sequences of tokens along with their word embedding numerical representation as input [23–25]. Recently, emphasis has been given to transformer-based pre-trained models, such as GPT and BERT [26–28], in which we have provided some contribution which will be integrated into the broader VM4SEC platform [29]. Despite their importance in security monitoring, no operationalized VPMs can be found neither in the literature nor on the market, either standalone or as part of existing security analysis tools and platforms.

For better understanding of the main novelty of the VM4SEC platform, in Table 1, a comparison between existing widely used static analysis platforms and our VM4SEC platform is provided. In particular, we compare the VM4SEC platform with SonarQube (https://www.sonarsource.com/products/sonarqube/, accessed on 5 June 2023), Coverity (https://scan.coverity.com/, accessed on 5 June 2023), and Fortify (https://www.microfocus.com/en-us/cyberres/application-security, accessed on 5 June 2023), which are popular static code analyzers known to be able to detect security issues.

**Table 1.** Comparison between existing popular static analysis tools and platforms and the VM4SEC platform.

| Requirements | SonarQube | Coverity | Fortify | VM4SEC |
|---|---|---|---|---|
| Identification of Security Issues | ✓ | ✓ | ✓ | ✓ |
| Visualization of Analysis Results | ✓ | ✓ | ✓ | ✓ |
| Quantitative Security Metrics | | | | ✓ |
| Vulnerability Prediction | | | | ✓ |

As can be seen by Table 1, existing ASA platforms provide identification of security-related bugs and visualization of the analysis results in a more meaningful and easy-to-understand way. However, none of the ASA platforms provide high-level security metrics, nor a quantitative estimation of the overall security of a given software application, which is important for security monitoring and decision-making during the overall software development lifecycle of a given software product. In addition to this, none of the existing platforms provide vulnerability prediction models, and, to the best of our knowledge, no operational VPM exists on the market which can be used directly for analyzing a given software application. To the best of our knowledge, VM4SEC is the only platform available that provides a high-level quantification of the security level of a given software product based on state-of-the-art software security evaluation models, as well as the only platform that provides ready-to-use text-mining-based VPMs. *It should be noted that since the VM4SEC is based on static analysis, the other three platforms could be used as input for the VM4SEC platform for providing its high-level security metrics, as well as for the VPMs. Hence, it can be also viewed as an extension of existing ASA platforms, providing additional and high-level functionality. In other words, the VM4SEC can be used complementarily with these static analysis platforms, in order to leverage their security issue identification capabilities and enhance their functionality by providing high-level security measures and vulnerability predictions.*

Finally, it should be noted that novel security monitoring solutions such as quantitative security assessment mechanisms and vulnerability prediction models require being properly configured and adapted in order to be utilized in practice. For instance, quantitative security assessment mechanisms should be calibrated properly in order to identify specific types of security issues and compute security metrics that are more relevant and of high importance for a specific company and/or application domain. Similarly, dedicated VPMs need to be built for the software projects on which they should be applied for monitoring their security level. This is a tedious process that prevents companies from using such solutions in practice. To the best of our knowledge, no other research work exists in the literature that presents how novel security solutions can be adapted to fit into the pipelines of specific companies and the workflows of specific development teams.

## 3. Methodology

In this section, a detailed description of the overall methodology that was followed for setting up a security monitoring platform tailored to the needs of a specific company is provided. The overall methodology is illustrated in Figure 1. As can be seen by Figure 1, the adopted approach consists of the following major steps:

1. **Survey:** Initially, a 2-step survey was conducted for identifying (i) the current state of the company with respect to employed security monitoring techniques, (ii) the most suitable security monitoring mechanisms that should be added to the workflows of its software engineers, and (iii) the most common security issues and most critical security aspects faced during the development of its software applications. The first two points are important for eliciting the requirements of the envisaged platform, whereas the latter is crucial for properly configuring the parameters of the platform's mechanisms, so as to be tailored to the company's needs. Online questionnaires were utilized as the main instruments for gathering the required feedback from the participants.
2. **Focus group**: Subsequently, an open discussion was conducted with the software engineers and project managers of the company, based on the topics discussed in the questionnaire, through a dedicated focus group. The open discussion provided additional insights about the current security state of the company, its security monitoring needs, and the most suitable security mechanisms to be deployed, along with their required configuration. The focus group was also vital for defining the requirements, the use cases, and, eventually, the final architecture of the VM4SEC platform.
3. **Requirements Elicitation and Architecture Design**: The feedback collected through the 2-step survey and the focus group was utilized as the basis for eliciting the functional and non-functional requirements of the platform, along with its main use cases. A formal requirements elicitation process was followed in order to ensure transparency and clear specification. Apart from the information collected through the survey and the focus group, several iterations of *question and feedback* sessions were conducted with the software engineers and the project managers of the company, in order to ensure the completeness of the requirements list. Based on the extracted requirements, the overall architecture of the platform was defined following the Data Flow Diagram (DFD) architectural design paradigm.
4. **Model/Mechanisms Configuration/Tuning**: The selected security monitoring mechanisms, i.e., the quantitative security assessment (QSA) and vulnerability prediction models (VPMs), were calibrated in order to adapt to the specific needs of the company. More specifically, as will be explained later, the VPMs were constructed based on a vulnerability dataset that was curated with the aid of the company's developers, comprising real-world vulnerability examples that are commonly found in the company's projects. Moreover, the QSA model was built is such a way so that it will be able to identify security issues that are considered more relevant and important for the applications of the company, as well as compute high-level metrics that are of interest to the company. For this purpose, information that was gathered by the questionnaire was leveraged and statistically analyzed, whereas advanced multi-criteria decision-making (MCDM) techniques were employed for deriving the models' parameters.
5. **Implementation and Deployment**: Following the microservices architectural paradigm, the mechanisms were implemented as standalone tools, while a unified user-friendly user interface (UI) is currently under development, aiming to enable the easy invocation of the security monitoring mechanisms and the visualization of the security evaluation results. Once the integrated VM4SEC platform reaches a fully functional level, it will be deployed on the premises of the company, in order to be evaluated in practice through its utilization during the company's development workflows.

## *3.1. Industrial Case Study*

### 3.1.1. Survey and Focus Group

The first step towards building a security monitoring platform tailored to the needs of a specific software company is to identify and report its actual needs. This can be achieved by consulting the employees of the actual targeted company in order to retrieve information about the security monitoring activities that they already employ during their SDLC, which security monitoring activities are considered more important to them, and what novel security monitoring mechanisms are more suitable to be added to their pipelines/workflows. This can be achieved through dedicated surveys, focus groups, and active communication with the company on a consultation basis.

In the present work, we have selected an actual company, namely Onelity, as a case study for demonstrating our approach. Onelity is an IT Services Provider working actively for leading Automotive, Telecommunications, Financial and e-Commerce organizations in Europe. It collaborates with its customers to turn their digital visions into results through its regional offices in Germany, Greece, and Cyprus. It was founded in 2020 by a team of highly qualified professionals, having more than 20 years of international experience in the field of Information Technology. It supports and provides customized turnkey solutions in mid- and long-scale projects all around Europe by using the latest technologies, and a full toolbox of frameworks and systems. It also provides the most advanced and up-to-date training programs in the market. At the time of writing, Onelity (hereafter referred to as *Company*) employs more than 50 highly skilled professionals.

In order to gather the required information, a survey and a focus group were conducted with employees of the Company. In particular, *16 key employees* of the Company were used as subjects of the study, acting as the participants both in our survey and in the focus group. We ensured that all of the selected participants are involved in the SDLC of the core software products that are developed by the Company from various roles, ranging from software developers to project managers. We also ensured that multiple people from each role would be involved in our study in order to further avoid potential bias.

As already stated, the information-gathering process was based on a survey and a focus group. These two information-gathering mechanisms are described in detail in what follows. In fact, we have followed the guidelines for empirical case studies provided by [40,41], in order to properly construct our survey and quantitatively analyze its results.

#### Two-Step Survey

Initially, a 2-step survey was conducted with the 16 participants of the Company, in order to gather the required information. The division of the survey into two sequential steps was necessary, as we first needed to understand the actual needs of the Company, i.e., which novel security monitoring mechanisms should be deployed into their pipelines, and based on the identified needs to gather dedicated information that is necessary for properly tuning/configuring the selected mechanisms to better fit into the Company's pipelines.

***Step 1:*** The purpose of the first step of the survey was to (i) identify the current state of the Company with respect to security monitoring, (ii) identify their needs with respect to further security monitoring activities that should be applied during their SDLC, and (iii) chose the most suitable security monitoring mechanisms that should be added to their pipelines/workflows. To gather this information, as will be explained later in more detail, a dedicated questionnaire was constructed and shared with the participants, as presented in Table 2. Initially, the first two parts of the questionnaire (i.e., Part 1 and Part 2 in Table 2) were constructed and shared with the participants. In brief, the following broader questions (which were mapped into more concrete questions in the actual questionnaire) had to be answered by the participants:

- *What is your role in the company and how many years of experience do you have?*
- *Does your company employ a secure SDLC in the projects that you are involved?*
- *What pro-active and re-active security testing and monitoring activities do you employ during the SDLC?*
- *Which is the most important security activity in which your company needs to invest more in the future?*

The responses to these questions were based on multiple choice answers in order to give the option to the participants to select among predefined answers. Since we wanted to provide freedom to the participants and gather as much useful information as possible from their side, most of these questions also provided the “Other” option to allow the participants to give a different answer from those provided, if they considered it necessary. This was decided because we did not want to risk limiting (or potentially directing) the participants’ responses to specific answers. In addition to this, it should be noted that the increased freedom in the responses was necessary for this step, as it allowed us to better design the rest of our survey and collect useful information for the requirements elicitation process in the next steps.

***Step 2:*** The purpose of the second step of the survey was to gather information that is necessary for properly tuning/configuring the most suitable security monitoring mechanisms (as identified based on the answers of the first step), in order to be tailored to the pipelines/workflows and needs of the Company. In particular, the responses of the first step of the survey were analyzed and the main security monitoring mechanisms were detected. As will be discussed later in the text, the participants showed great interest in security monitoring solutions that are based on static analysis, particularly on quantitative security assessment (QSA) and vulnerability prediction models (VPMs). Hence, the questionnaire (see Table 2) was updated by adding two additional parts (i.e., Part 3 and Part 4) with questions necessary for tuning these models in the future. The broader questions that were asked in these two sections are presented below:

- *How important is the characteristic of ≪Security_Characteristic≫ for a software application (compared to the other characteristics)?*
- *According to your expertise, which Security Characteristics are significantly affected by ≪Security_Issue≫?*

The purpose of these questions is (i) to identify the main security aspects (i.e., characteristics) that are of high interest for the projects that are developed by the Company, as well as their relative importance, and (ii) to identify the main security issues that they face in these projects along with their impacts on important security aspects. In contrast to the answers to the questions of Step 1, which offered much freedom to the participants, the answers to the questions of Step 2 were provided either on a 5-point Likert Scale or based on a list of predefined choices. The answers in this step had to be strictly defined since (i) we are referring to official security terms, and (ii) the responses will be used for configuring the parameters of the models (see Section 3.3). Therefore, the consistency and correctness of the responses need to be ensured. The selected security characteristics are retrieved from ISO/IEC 25010 [42], whereas the Security Issues are retrieved from NIST’s Common Weakness Enumeration (CWE) (https://cwe.mitre.org/top25/) database. The values of the ≪Security_Characteristic≫ and ≪Security_Issue≫ that were selected are shown in Table 2, along with the summary of the final questionnaire.

**Questionnaire:** As already stated, as an instrument for gathering the required information from the 2-step survey, we opted for a questionnaire. The most important part of developing a questionnaire is the selection of questions. In our survey, this process was governed by the guidelines provided by Kitchenham and Pfleeger [43]: (a) keep the number of questions low, (b) questions should be purposeful and concrete, (c) answer categories should be mutually exclusive, and (d) the number, the order, and the wording of questions should avoid biasing the respondent. To this end, we constructed a questionnaire with 25 questions, organized into four parts (see Table 2).

As already stated, the first two parts were used in the first step of the survey, whereas the latter two were used in the second step of the survey. The questionnaire containing only the first two parts was initially distributed to the participants, who were asked to provide their answers. An initial analysis of their answers was conducted and the main security monitoring mechanisms that should be deployed regarding the SDLC of the Company product were identified. Then, the questionnaire was updated by adding the remaining two parts (i.e., Part 3 and Part 4), including questions specifically crafted for collecting information that is required for configuring/tuning the selected security monitoring mechanisms (see Section 3.3). The updated questionnaire was distributed to the participants, who were asked to provide answers to the questions of the remaining two sections. It should be noted that in the beginning of each section, the questionnaire introduced the participant to the involved security terms, in order to ensure that the participants had a clear understanding of the included terms and the questions.

**Table 2.** The questionnaire that was utilized for collecting data from the 2-step survey.

| ID | Question |
|---|---|
| | **Part 1—Demographics** |
| **Q1.1** | What is your role in the company? |
| **Q1.2** | How many years of experience do you have in this position? |
| | **Part 2—Adopted Security Activities and Further Needs** |
| **Q2.1** | What type of security activities do you utilize during the development of software products within your company? |
| **Q2.2** | What re-active security countermeasures do you apply during the development of software products in your company? |
| **Q2.3** | What pro-active security countermeasures do you apply during the development of software products in your company? |
| **Q2.4** | Which of the following security countermeasures do you consider critical and that the company should invest more on them in the future? |
| **Q2.5** | Which of the following novel security monitoring activities do you consider interesting and with practical value in order to be included in their pipelines? |
| | **Part 3—Important Security Aspects** |
| **Q3.1** | How important is the characteristic of Confidentiality for a Software application (compared to the other characteristics)? |
| **Q3.2** | How important is the characteristic of Integrity for a Software application (compared to the other characteristics)? |
| **Q3.3** | How important is the characteristic of Availability for a Software application (compared to the other characteristics)? |
| **Q3.4** | How important is the characteristic of Authentication for a Software application (compared to the other characteristics)? |
| **Q3.5** | How important is the characteristic of Authorization for a Software application (compared to the other characteristics)? |
| **Q3.6** | How important is the characteristic of Security Compliance for a Software application (compared to the other characteristics)? |
| | **Part 4—Critical Security Issues** |
| **Q4.1** | According to your expertise, which Security Characteristics are significantly affected by Injection Issues? |
| **Q4.2** | According to your expertise, which Security Characteristics are significantly affected by Integer/Buffer Overflow Issues? |
| **Q4.3** | According to your expertise, which Security Characteristics are significantly affected by Cross-site Scripting (XSS)? |
| **Q4.4** | According to your expertise, which Security Characteristics are significantly affected by Null Pointer Dereference? |
| **Q4.5** | According to your expertise, which Security Characteristics are significantly affected by Weak Cryptography Issues? |
| **Q4.6** | According to your expertise, which Security Characteristics are significantly affected by Security Misconfiguration Issues? |
| **Q4.7** | According to your expertise, which Security Characteristics are significantly affected by Insufficient Logging? |
| **Q4.8** | According to your expertise, which Security Characteristics are significantly affected by the utilization of Vulnerable and Outdated Components? |
| **Q4.9** | According to your expertise, which Security Characteristics are significantly affected by Server-side Request Forgery (SSRF)? |
| **Q4.10** | According to your expertise, which Security Characteristics are significantly affected by Broken Authentication Issues? |
| **Q4.11** | According to your expertise, which Security Characteristics are significantly affected by Insecure Deserialization Issues? |

Focus Group

After gathering information from the participants through the 2-step survey, a focus group was organized. The purpose of the focus group was to initiate an open discussion with the participants, based mainly on the contents of the questionnaire and their responses, in order to gain a better understanding of their view with respect to the discussed topics and further insights that could not be gathered from a single questionnaire. It also enabled us to ask follow-up questions, driven by the discussion, which could lead to additional information that may be highly useful for understanding the actual needs of the Company and for better defining the envisaged security monitoring platform. Hence, the 2-step survey was crucial for the effective design and organization of the focus group.

Similarly to the 2-step survey, we followed the guidelines of [41,43] for conducting industrial case studies and analyzing their results. In particular, the formal approach that we followed for planning, designing, and conducting the focus group is summarized below:

- During the ***planning*** of the focus group we defined four major goals, *"to discuss (a) the current state of the Company with respect to security monitoring; (b) the security monitoring needs of the Company; (c) the desired functionalities and qualities that a security monitoring platform must exhibit; and (d) the most important security aspects and the most critical security issues of their developed software projects"*.
- Regarding the ***design***, as can be seen in Table 3, the focus group was divided into five blocks (i.e., parts), each one focusing on a separate topic. Each one of the first four blocks (Block 1–Block 4) was dedicated to each one of the four main goals of the focus group, whereas the last block (Block 5) was defined so as to satisfy the communication protocol. The focus group was intended to last for 1 h in total and to be conducted using a teleconference platform. Each one of the main blocks (i.e., Blocks 1 to 4) was expected to last approximately 14 min, whereas the last block, which was about closing the focus group, was expected to last 4 min.
- While ***conducting*** the focus group, the discussion was focused on the aforementioned discussion axes, which are reflected by the blocks presented in Table 3. It should be noted that in many cases when there was an agreement among the participants, we asked the remaining participants to only provide complementary or contradictory claims. With respect to Blocks 1–4, we asked the participants to consider the core software products in which they are actively involved during their development when providing their answers. This would lead us to more representative results and better tuning of the security monitoring mechanisms.

It should be noted that the focus group, apart from additional insights on the topics and questions asked in the dedicated questionnaire, provides vital information that is necessary for eliciting the requirements and designing the architecture of the envisaged security monitoring platform (see Section 3.2). As can be seen in Table 3, the third block of the focus group allowed us to have an open discussion about the main functionalities and the quality attributes that the envisaged platform should exhibit, in order to be considered useful and practical by the participants. Hence, as will be discussed in Section 3.2, this information was important for the overall design of the VM4SEC platform.

### Core Findings

In the present section, we provide the core findings of the previously described two-step survey and focus group. For reasons of brevity, only the main findings that were considered important for the design and development of the VM4SEC platform are presented and summarized.

In Figures 2 and 3, we provide the demographics of the participants that were involved both in the two-step survey and in the focus group, as gathered by the questionnaire. As can be seen in Figure 2, the vast majority of the participants (i.e., 50%) were actual developers, followed by Quality Assurance (QA) Engineers (i.e., 19%), 13% were project managers, and the remaining percentage (i.e., 6%) was equally distributed among software engineers, software architects, and DevOps engineers. Hence, we have representatives from the whole SDLC, whereas the main body of the responses stems from people that are actively involved in the actual development of the software projects. In addition to this, as can be seen in Figure 3, 56.3% of the participants have less than 5 years of working experience, while 12.5% have more than 10 years of working experience.

**Table 3.** Focus Group Topics.

| Focus Group Structure |
|---|
| **Block 1: Current state of security of the company** • Do you use re-active security countermeasures in your software projects? If yes, which are the most commonly used? • Do you used pro-active security testing/monitoring mechanisms? If yes, which are the most commonly used? • Do you prefer static or dynamic security testing and why? Are there any shortcomings with those techniques that you would like to improve? |
| **Block 2: Security Monitoring Needs** • Are you aware of any novel security monitoring and optimization solution that you believe that it would be worth being added to your pipeline? • Do you think that having quantitative security metrics is useful? • Do you think that having vulnerability prediction is useful? • Do you think that having ML-based fuzz testing is useful? • Do you think that having ML-based penetration testing is useful? |
| **Block 3: Security Monitoring Platform Functionality and Usability** • What functionalities should a security monitoring platform have? • What additional features should be provided in order to improve the usability and the practicality of the security monitoring platform? • What qualities should the platform exhibit? (e.g., number of concurent users, acceptable down time) • What are the most useful scenarios that you can think of? |
| **Block 4: Parameterization of the selected monitoring mechanisms** • Which security characteristics are more important for your projects? • Which security issues are more critical and commonly found in your software projects? • Which of these issues affect each one of the security characteristics of interest? |
| **Block 5: Conclusion** • Thank you for your time • Explain next steps (. . .) • Ask if they would like to receive results by email. |

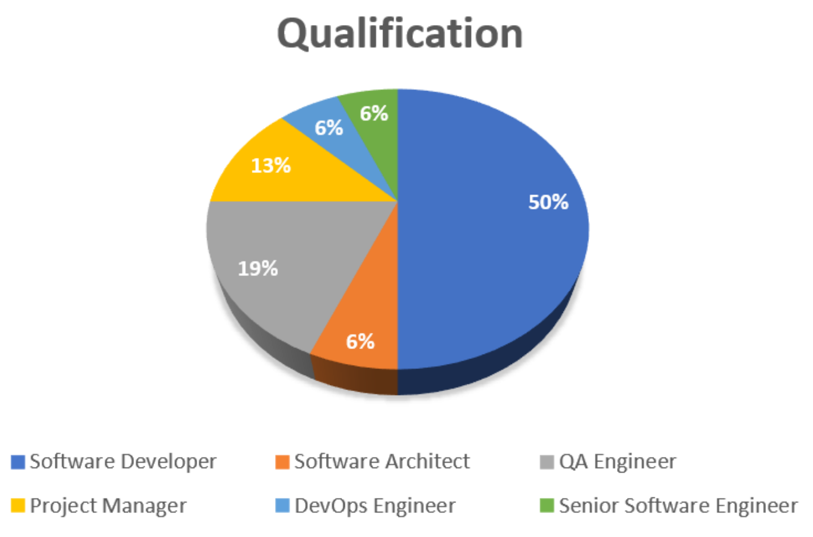


**Figure 2.** Industrial Case Study Demographics—Participants' Role in the Company.

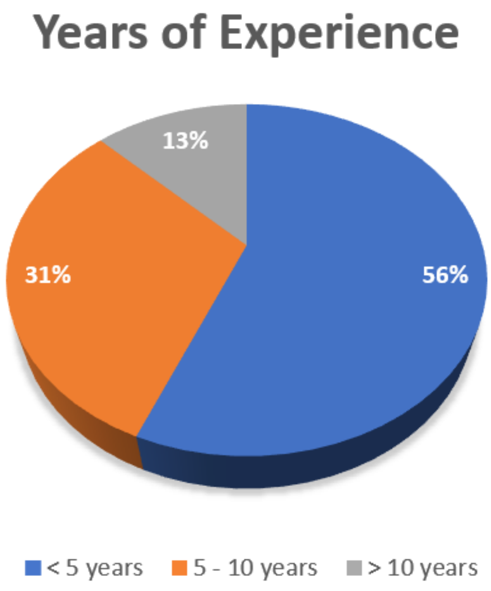


**Figure 3.** Industrial Case Study Demographics—Participants' Years of Experience.

**SURVEY:** Figure 4 presents the results of Q2.1 regarding the type of security testing (i.e., pro-active or re-active) the participants use during their SDLC. As can be seen, the vast majority of the participants stated that they use security testing approaches in their projects in order to enhance security. Among the re-active approaches (Q2.2), vulnerability patching and installation of firewall are the most widely used, followed by attack detection techniques and honeypots, as shown in Figure 5. An interesting observation is that around 31% of the respondents said that they do not normally apply re-active approaches in their projects. This was an engaging finding that was marked as a point for discussion in the focus group, in order to better understand the reasons why in some projects no re-active approaches are adopted. As will be discussed later in the focus group, these participants stated that they did not use re-active security approaches, as the software products that were working on did not have security concerns, and therefore the installation of security countermeasures was not considered necessary.

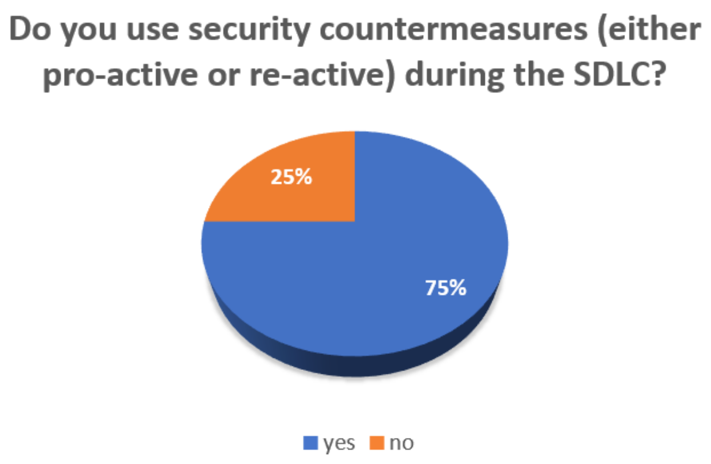


**Figure 4.** Adoption of Security Mechanisms by the Company.

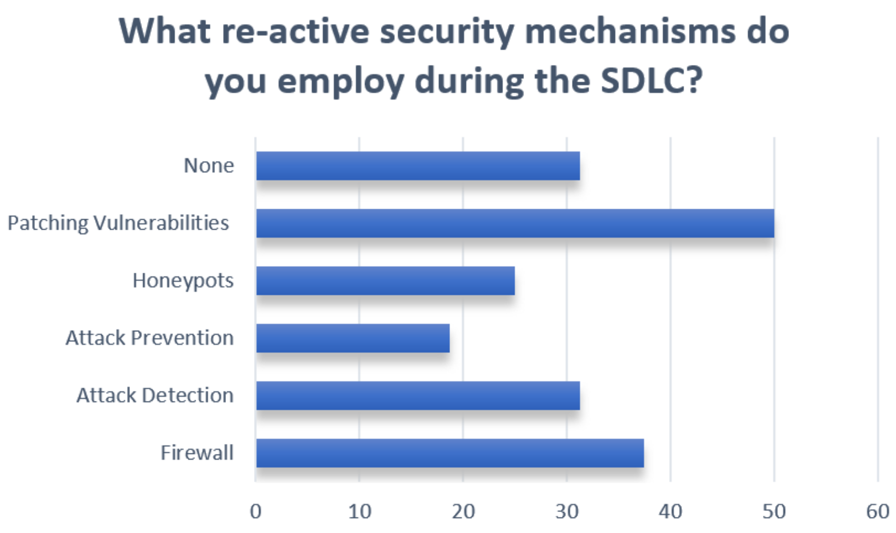


**Figure 5.** Adoption of Re-active Security Mechanisms by the Company.

In Figure 6, the results of Q2.3 with respect to the kind of pro-active approaches used by the Company during the SDLC are presented. As can be seen, around 69% of the participants stated that they employ dynamic security testing, followed by static testing, which was selected by around 31% of the participants. Around 19% of the participants stated that they do not utilize any pro-active security testing approach, either static or dynamic, during the overall development process. Similarly to Q2.2, as revealed during the focus group, the reason for not using pro-active security testing approaches was that the referred software products were not security-critical.

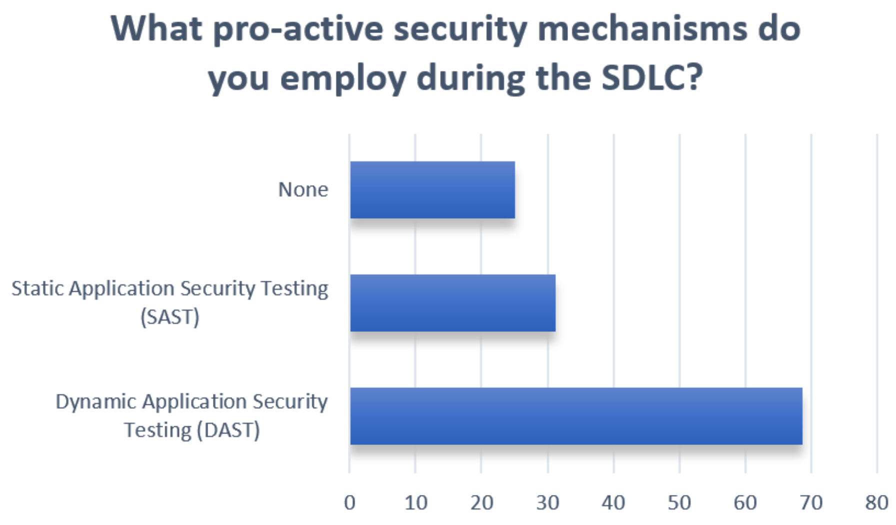


**Figure 6.** Adoption of Pro-active Security Mechanisms by the Company.

In Q2.4, the participants were asked to declare which one of the pro-active security approaches they consider useful, and therefore should receive more attention from the development team. As can be seen in Figure 7, around 75% of the participants stated that static analysis is considered the most promising security testing technique during the coding phase and therefore, deserves more attention from the Company. This contradicts the low utilization of static analysis for security purposes by the Company, which was observed in the responses of Q2.3. An interesting topic for discussion, which was left for the focus group, was to understand whether the participants selected static analysis because they find it indeed useful and helpful for adding security to their systems, or because they recognize that it is not widely used during their pipelines.

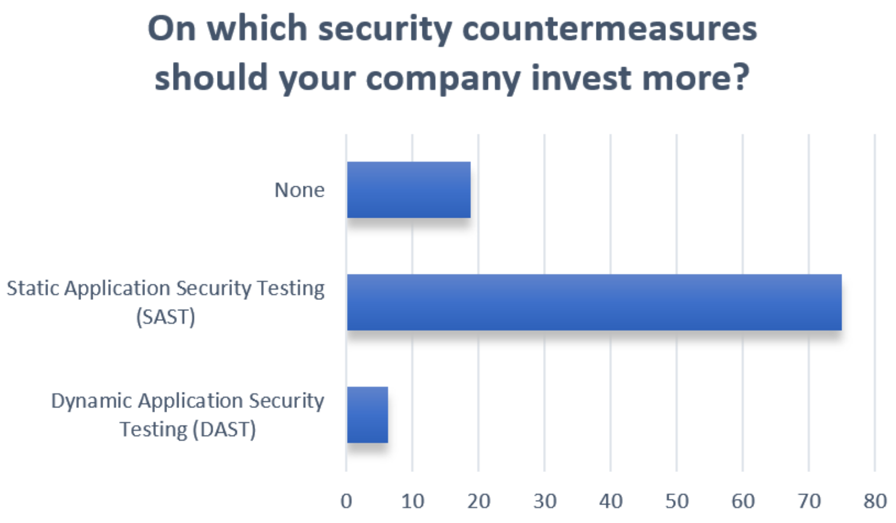


**Figure 7.** Pro-active Security Mechanisms of High Interest for the Company.

In Q2.5, the participants were asked which novel security monitoring mechanisms (from a given set) they consider to be interesting and with practical value, in order to be included in their pipelines. A summary of their responses is illustrated in Figure 8. As can be seen, the static-analysis-based security monitoring mechanisms, namely the quantitative security assessment (QSA) and the vulnerability prediction models (VPMs), were recognized as the most valuable ones, taking up around 88% and 94% of the votes, respectively.

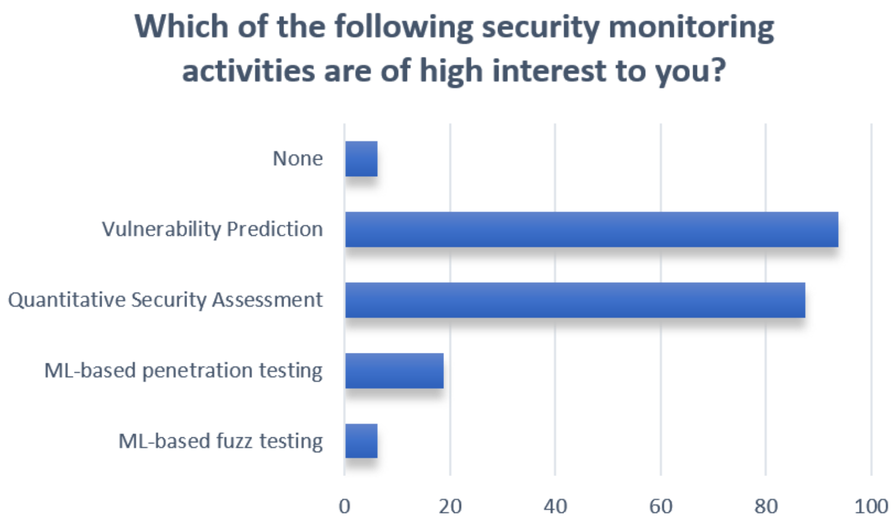


**Figure 8.** Novel Security Monitoring Solutions that are of High Interest for the Company.

By analyzing the responses presented above, we reached the conclusion that the participants are more interested in security monitoring mechanisms that are based on static code analysis, and particularly on the QSA and VPMs mechanisms. Based on this observation, as already discussed in "Two-Step Survey", the questions of Part 3 and Part 4 of the questionnaire were properly defined (see Table 2), in order to collect information necessary for configuring/tuning those mechanisms.

For reasons of brevity, we provide the main findings of these questions. According to the participants' responses, the security characteristics of *Confidentiality*, *Availability*, and *Integrity* were considered the most important security aspects of their software products. In addition to this, the security issues that greatly affect each one of these security characteristics was identified. The most critical security issues that were identified were the *Null Pointer* references.

More information on how these results were leveraged is provided in Section 3.3, where the models are constructed. As can be seen also by inspecting Table 2, the questions of Part 3 and Part 4 of the questionnaire are meant for gathering statistics, which would be further processed by us in order to configure the core security monitoring mechanisms that were selected and described in Section 3.3. Illustrating the "raw" charts would take up much space without providing any added value to the discussion.

**FOCUS GROUP:** As already stated, the survey was followed by a focus group. As made clear by the above description, many interesting observations were made during the analysis of the responses of the questionnaire that led to additional questions that had to be further discussed in order to gain more insight. Based on the process described previously, the focus group was conducted. The main observations from each block of the focus group are presented in what follows.

**Block 1:** The vast majority of the participants stated that they prefer static analysis over dynamic analysis for security testing. This is in line with the results of the first step of the survey, in which static code analysis was recognized by the participants as one of the most interesting security testing activities that they should employ during the SDLC (see Figure 7). When asked why they prefer static analysis, the most common reason was its ability to highlight a security issue along with its location in the source code, followed by the high automation of the approach and its ability to be applied even before the code can be executed or even compiled. This is in line with the results of other popular surveys on the usefulness of ASA [3]. When asked about the shortcomings of static analysis, the main shortcoming that was reported was the large volume of alerts that it produces, which is often difficult to manage. Equally important was the lack of interpretation of the results. The participants, especially the project managers, expressed the difficulty that they face in understanding the security information that resides in these alerts, due to the fact that they are in a raw format. This is the main reason for the limited adoption of such approaches in practice, despite their acknowledged benefits. All the participants agreed that post-processing tools able to extract useful information from the raw alerts produced by static analysis are highly useful and of practical importance.

It should also be noted that the results of the focus group were in line with the results of the survey with respect to the applied re-active and pro-active approaches. In fact, security patching and firewalls were the most widely used re-active approaches; whereas, with respect to the pro-active approaches, dynamic testing was more frequently used than static testing. Those participants that said that they do not use security testing approaches, were actually working on software applications with no security considerations.

**Block 2:** None of the participants were fully aware of the discussed trends in the field of software security monitoring. Hence, the discussion was then directed to the four highly popular novel security monitoring/testing techniques, two from dynamic and two from static analysis (see Table 3). A brief description of each one of those four mechanisms was provided in order to ensure that the participants had a sufficient understanding of its purpose and functionality. The vast majority of the participants expressed their interest in the ***quantitative security assessment*** and ***vulnerability prediction models***, since they do not have similar tools in their pipeline and they think that they provide useful insight during the development. In addition to this, the majority of the participants did not consider the utilization of ML-based fuzzing and penetration testing useful for their pipelines, since they already apply common fuzzing and penetration testing tools and they consider them already accurate and sufficient. These outcomes are in line with the answers that they provided during the survey through the questionnaires. However, the focus group allowed us to realize the reasoning behind this selection. In particular, this explains the reason why in Q2.4 the participants ranked dynamic security testing so low, as they consider it a traditional approach that is already part of the process, without additional interest.

**Block 3:** Apart from the core functionalities, the platform should also provide additional features that are considered to be important and useful by the software engineers. The questions of the third block of the focus group helped us in identifying these requirements. In brief, the participants (in fact, the software engineers that are involved in the development of the Company's software) commonly expressed the need for the platform to access the source code directly from version control systems such as GitHub, Bitbucket, and GitLab, as these are the repositories in which their projects reside. In addition to this, there was a consensus on the need for a GUI able to visualize the results. To this end, several non-functional requirements were determined, such as acceptable downtime, analysis speed, etc. The information collected through the focus group was highly useful for the requirements elicitation and use case definition of the VM4SEC platform, a process described in the next section.

**Block 4:** In the fourth part of the focus group, emphasis was given to the main security aspects/characteristics of the software applications that are actively developed by the company, the security issues that they normally face, and how these issues affect the various security characteristics. All of the participants agreed that the characteristics of Confidentiality, Integrity, and Availability are the most critical security aspects that should be satisfied at minimum by any software project under development. This is in line with what was observed in the third part of the questionnaire. Then, for each one of these three security aspects, the most critical security issues that may affect them were identified. The identified critical issues are in line with what was retrieved by the questionnaire, further enhancing our confidence with respect to the reliability of the responses, and, in turn, to the correctness of the final model parameters. Minor inconsistencies were discussed and appropriate updates were made to the ranked list when necessary.

**Main Takeaways:** In summary, through the discussion carried out during the focus group and the responses that were provided through the questionnaires, we observed that the employees of the Company consider static analysis as an important mechanism for identifying security issues during the development process. We also identified a need for novel mechanisms able to post-process the static analysis results in order to extract security-related information that is encapsulated in them. Finally, the participants believe that they sufficiently cover dynamic security testing through penetration testing and fuzzing, and that the utilization of ML to further improve them is not critical. Among the presented novel security mechanisms, static-analysis-based quantitative security assessment and vulnerability prediction were found to be the most interesting mechanisms that they would like to add to their pipelines.

To this end, we decided to build a static-analysis-based security monitoring platform, namely the *VM4SEC platform*, being able to post-process the results of static analysis in order to conduct quantitative security assessment and vulnerability prediction.

At this point, a remark about the importance of the focus group is considered necessary. From the above analysis, it is clear that the focus group helped us to better understand the reasoning behind the responses of the participants, and gain better insights into the current state and needs of the Company. For instance, we were able to understand how important static analysis is considered by the employees of the Company, along with their willingness to utilize it more actively in their workflows via mechanisms that will improve their experience with static analysis tools and help them gain deeper insight from the raw static analysis results. We also realized that the low interest in investing in dynamic security testing techniques that were reported in the questionnaires was not due to the fact that they consider them less important, but because they have already deployed such tools in their pipeline, they consider them mature enough and there is no need for further investing in dynamic testing. Making these observations would not be possible by simply analyzing the responses of the questionnaires, which could have led us to wrong conclusions. Hence, we consider a follow-up focus group a necessity for gaining correct feedback from the employees.

### 3.2. Requirements Elicitation and Architecture Design

From the survey and the focus group presented in Section 3.1.1, we collected valuable feedback from the Company with respect to its actual needs in terms of security monitoring. This feedback formed the basis for eliciting the requirements (i.e., both functional and non-functional) and the use cases of the VM4SEC platform, and, in turn, for defining its overall architecture and technical specifications. It should be noted that in addition to the survey and focus group, a close communication channel was established with the software engineers and project managers of the Company, and several bilateral discussions were conducted during the requirements elicitation and architectural design process, in order to ensure that the elicited requirements, the defined use cases, and the final architecture accurately reflect the needs of the Company. In the present section, we provide a description of the approach that was followed for eliciting the requirements of the platform and deriving its final architecture based on these requirements.

#### 3.2.1. Requirements Elicitation and Use Case Definition

For eliciting the requirements of the envisaged VM4SEC security monitoring platform and defining its main use cases, we followed a formal software engineering methodology, which is based on the results of empirical/industrial studies. It is a methodology that we have widely used in past EU and nationally funded research projects (e.g., SDK4ED and IoTAC) for requirements elicitation, based on formal principles and mature techniques from the software engineering community. The whole requirements elicitation and use case definition methodology that we followed is illustrated in Figure 9.

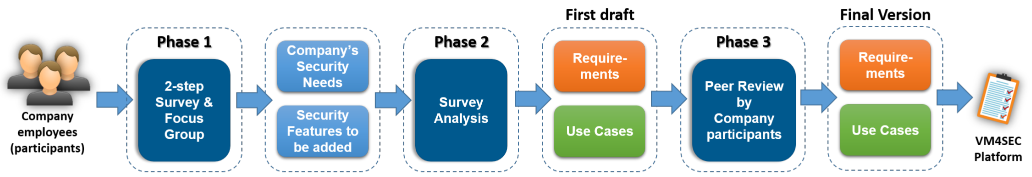


**Figure 9.** The high-level overview of the adopted requirements elicitation and use cases definition process.

As can be seen in Figure 9, the whole process consists of three main phases. In the first phase, the industrial study that was described in detail in Section 3.1.1 was conducted, which retrieved information from the subject Company through a two-step survey and a focus group. It allowed us to identify the main security monitoring needs of the Company, the core functionalities that the platform should provide, and the additional features that it should exhibit in order to be useful and practical. In the second phase, we analyzed the main outcomes that were collected by the empirical study and extracted a set of requirements that reflect the expressed needs and their corresponding use cases. In the third phase, the participants that took part in the case study performed a peer review of the produced list of the defined requirements and use cases, verified the correctness and completeness of the list, and proposed corrections and additions. Based on their feedback, the initially defined requirements and use cases were finalized. This led to the final list of requirements and use cases that the envisaged platform should satisfy. In particular, the aforementioned process led to 20 requirements and 5 use cases.

More specifically, based on the information collected from the empirical case study described in Section 3.1.1 and the process presented in Figure 9, 20 requirements (8 functional and 12 non-functional) were defined. The requirements were defined using common and widely used formal templates. Based on our experience, enforcing the utilization of templates for the formal representation of the requirements allows the finally defined requirements to be (i) more understandable; (ii) easy to follow and inspect, as a unified approach is used for their description; and (iii) less error-prone. For the construction of the requirements, we adopted the guidelines of the ideal functional requirement construction enunciated in the standard IEEE 29148-2011. According to this standard, the definition of the functional requirements should have the following attributes (as a minimum):

- An *ID* for uniquely identifying the requirement, allowing it to be referenced without having to use its complete name.
- A *Priority* for declaring how important is the defined requirement for the overall system, in order to be given higher priority during the development. For defining the priority of the requirements, the Moscow method [44] was used. The priority field takes an integer value between 1 and 4, with 1 corresponding to the highest priority (i.e., declares a must-have feature) and 4 to the lowest priority (i.e., declares a secondary/optional feature).
- A *Category* that classifies the requirement to a specific group. This is important for grouping the requirements into closely related categories that should be considered together.
- A *Dependency* field for denoting whether the defined requirement depends on other requirements of the system. In this field, the IDs of the requirements on which it depends should be provided.
- A *Short Description* field that contains a very brief description of the defined requirement. This description should be clear and concise. At minimum it can be the name of the defined requirement.
- A *Long Description* field that contains an extended description of the defined requirement. The long description should supplement the information that is provided in the Short Description field, in order to facilitate the understanding of the defined requirement.
- A *Rationale* field that contains an explanation of why the defined requirement is necessary and important for the broader system.
- A *Condition* field that contains a description of the pre-conditions that should be satisfied in order for the defined requirement to be valid.
- An *Expected Input* field that contains a description of the inputs (if any) that are required by the defined requirement.
- An *Expected Output* field that contains a description of the outputs (if any) that are produced by the system and are relevant to the defined requirement.
- An *Expected User Interface* field that contains a description of the desired format in which the outputs of the requirements should be presented to the user.

Based on the above template, the requirements were defined in a tabular form, which makes them more readable and understandable. In Tables 4–6 we provide three core functional requirements that we derived based on the process defined in Figure 9 by utilizing the aforementioned template.

After eliciting the requirements of the VM4SEC platform, we defined the main use cases, which actually correspond to the main usage scenarios of the envisaged platform. Again, based on the industrial case study presented in Section 3.1.1 and the overall process illustrated in Figure 9, we defined six use cases. Similarly to the requirements definition step, we decided to use a formal template for the definition of the use cases, in order to enhance their clarity and ensure their understandability both by the participants and by the developers of the VM4SEC platform. The template that can be used to present the use cases is based on the guidelines introduced by Cockburn [45]. The template provides the following entries for each use case:

- An *ID* to identify univocally the use case. The format of the ID could be similar to the one of the functional requirements, in order for the overall requirements and use case management to be easier.
- A *Short Description* to shortly and uniquely describe the use case with a verb phrase (the goal of the primary actor).
- The *Frequency* at which the use case is expected to happen.

- The *Scope* that reports what system considers black-box. It could be one of the following:
  - System: the use case refers to the system as a whole.
  - System functionality: the use case refers to an individual functionality of the system.
- The *Priority* of the use case, in terms of criticality, to the envisioned platform.
- A *Long Description* to further describe the use case, if needed.
- A *Related Functional Requirements* field containing the IDs of the Functional Requirements (FRs) that specify the detailed functionality involved in the use case.

**Table 4.** Functional requirements for identification of software security issues.

| | |
|---|---|
| **ID** | FR 1 |
| **Priority** | 1 |
| **Category** | Software Security Monitoring |
| **Dependence** | None |
| **Brief Description** | Identification of software security issues |
| **Detailed Description** | The system must detect security issues (i.e., potential vulnerabilities) that reside in the source code of the analyzed software application and report the identified issues to the user. |
| **Rationale** | Vulnerabilities usually stem from mistakes that are made by the developers during the coding phase of the SDLC. The ability to identify such security issues is critical for the optimization of the security level of the produced source code. |
| **Condition** | Access must be granted to the source code of the software application that should be analyzed. |
| **Expected Input** | The source code files of the selected software application. |
| **Expected Output** | A complete list of all the potential security issues that were identified, along with recommendations and examples of potential fixes (where possible). |
| **Expected Interface** | Integration to the broader platform and presentation of the results through visualization entities (e.g., charts and tables). |

**Table 5.** Functional requirements for assessment of the security level of the software.

| | |
|---|---|
| **ID** | FR 3 |
| **Priority** | 1 |
| **Category** | Software Security Assurance |
| **Dependence** | FR 1 |
| **Brief Description** | Assessment of the security level of the software |
| **Detailed Description** | The system should assess the level of internal security of a software application whose code has been analyzed for potential vulnerabilities, providing quantitative security indicators. |
| **Rationale** | This is expected to facilitate decision-making during the implementation of software applications by the development team by providing a quantitative measurement of their security level. |
| **Condition** | To have successfully performed static code analysis. |
| **Expected Input** | The source code files of the software application. |
| **Expected Output** | A report containing both the overall security score of the analyzed software application and the individual lower-level security property scores, as well as the category in which the problem is classified. |
| **Expected Interface** | A table reporting the overall security index, as well as various graphs illustrating security properties at lower levels of a software product. |

**Table 6.** Functional requirements for software vulnerability prediction.

| | |
|---|---|
| **ID** | FR 4 |
| **Priority** | 1 |
| **Category** | Software Security Assurance |
| **Dependence** | FR 5 |
| **Brief Description** | Software vulnerability prediction |
| **Detailed Description** | The system must identify the vulnerable components (i.e., classes, methods, etc.) in a software product based on the existing code and the results of the machine learning model that is trained on the knowledge base. It should provide a categorization so that the software development team can focus on the software components that have an increased likelihood of being vulnerable. |
| **Rationale** | The information provided by implemented vulnerability prediction models can be used by developers to prioritize their efforts to address security-related issues in software, thus allocating the limited testing resources to higher risk areas (i.e., vulnerable components). |
| **Condition** | To have access to the source code of the software under analysis. |
| **Expected Input** | The source code files of the software application. |
| **Expected Output** | List of software components that are likely to be vulnerable. |
| **Expected Interface** | AA graphical representation of the identified components, along with detailed information generated by the analysis. |

Based on the above template, the use cases of the VM4SEC platform were formally defined. In Tables 7 and 8 are presented the two core use cases of the platform, which correspond to the two main security monitoring mechanisms that the platform should provide according to the Company's feedback, namely quantitative security assessment and vulnerability prediction.

**Table 7.** Use case about performing quantitative software security assessment.

| | |
|---|---|
| **ID** | Use Case 2 |
| **Brief Description** | Performing quantitative software security assessment |
| **Frequency** | Several times a day upon user request. |
| **Main User** | User |
| **Scope** | System Functionality |
| **Priority** | 1 |
| **Detailed Description** | This use case describes how a user can assess the overall security level of a selected software application. |
| **Related Functional Requirements** | FR 1, FR 2, FR 3, FR 5 |
| **Basic Flow** | |
| **Step 1 (Actor: User)** | The user declares that he/she wishes to assess the security level of a selected software application. The user also declares (optionally) whether he/she wishes to see the criticality of the identified security issues. |
| **Step 2 (Actor: System)** | The system statically analyzes the source code of the selected software application and runs the implemented Security Assessment model to calculate the overall security score. |
| **Step 3 (Actor: System)** | Finally, the system displays a report to the user with the detailed results of the security assessment. |

**Table 8.** Use case about executing the software vulnerability prediction model.

| | |
|---|---|
| **ID** | Use Case 3 |
| **Brief Description** | Executing the software vulnerability prediction model |
| **Frequency** | Several times a day upon user request. |
| **Main User** | User |
| **Scope** | System Functionality |
| **Priority** | 1 |
| **Detailed Description** | This use case describes how a user can identify potentially vulnerable software components of a selected application. |
| **Related Functional Requirements** | FR 1, FR 2, FR 4, FR 5 |
| **Basic Flow** | |
| **Step 1 (Actor: User)** | The user declares that he/she wishes to be informed about the components of the selected software application that may be vulnerable. |
| **Step 2 (Actor: System)** | The system analyzes the source code of the selected software product and runs the implemented vulnerability prediction model to categorize the software components as vulnerable or not. |
| **Step 3 (Actor: System)** | Finally, the system displays to the user a report of the software components that have been predicted to be vulnerable, together with a score reflecting the likelihood that they are vulnerable. |

### 3.2.2. Architectural Design

After devising the requirements and the use cases of the VM4SEC platform, we proceeded with the design of its overall architecture. For the architectural design of the VM4SEC platform, we followed the guidelines of the IEEE 1471 international standard. According to this standard, starting from the functional requirements and the use cases of a system, three main viewpoints of its architectures can be defined, which are the logical, the functional, and the deployment viewpoints.

As our main architectural design technique, we have selected the *Data Flow Diagram (DFD)* technique, which provides a convenient graphical means for analyzing the structure of the system at different levels of abstraction, focusing on the flow of data among internal and external entities [46]. According to Avison et al. [47], the DFD technique is one of the most effective techniques for visualizing the flows and the processing of data within a software system. The DFD technique is based on the hierarchical decomposition of the system following a top-down approach [46], providing in that way several levels of abstractions for the given system.

In the highest level of abstraction, which is known as the context diagram or the Level-0 DFD, the system is represented as a black box, showcasing its interconnection with external entities (e.g., users and third-party systems). In fact, the context diagram showcases the boundaries of the system, separating it from external entities and highlighting its input and output flows. The context diagram of the envisaged VM4SEC platform, as specified based on the collected information (see Section 3.1.1) and the guidelines of the IEEE 1471 standard is illustrated in Figure 10.

As can be seen in Figure 10, the context diagram is useful to (i) visualize the scope of the system, (ii) highlight the external entities that interact with the system, and (iii) summarize the input and output flows of the system. More specifically, as can be seen in Figure 10, the VM4SEC platform has five external entities, one of them being the actual user of the system, and the other four being external tools that are required for retrieving the source code of the selected software application (i.e., Version Control System) and analyzing it in an attempt to detect security issues and vulnerability patterns (i.e., Static Code Analyzers, Text mining tools, and Software Metrics Tools).

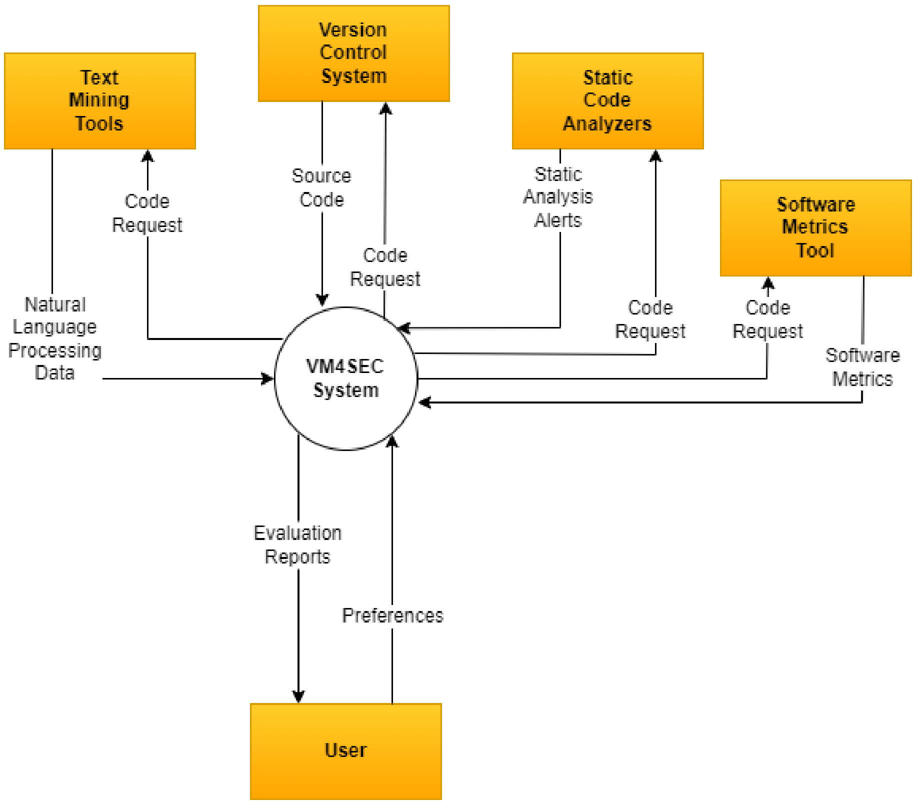


**Figure 10.** The context diagram of the VM4SEC security monitoring platform

As already stated, the DFD technique is based on the hierarchical decomposition of the system. Hence, the context diagram was further elaborated and decomposed into Level-1 and Level-2 diagrams, which were useful for the VM4SEC component identification and specification. These diagrams provided an overview of the main entities of each feature of the VM4SEC platform, their main inputs and outputs, and their main data processing activities that are performed. In Figure 11, the Level-1 DFD of the VM4SEC platform is illustrated, which showcases the core features of the overall platform, namely "Quantitative Security Assessment", "Vulnerability Prediction", and "Report Generation".

As can be seen in Figure 11, the Level-1 DFD allows the user to understand which external entities must be invoked for performing the associated functionality, what data are produced, and which data are displayed to the user. For instance, as shown in Figure 11, the quantitative security assessment entity receives the source code of the selected software application from a version control system, requests the source code to be analyzed by external static code analyzers and software metrics tools, and, based on a pre-stored security assessment model, performs a security analysis, the results of which are stored into a dedicated data store named security analysis results. Then, a report generation entity takes those assessment results and turns them into visual reports, which are displayed to the user. A similar procedure is followed for the vulnerability prediction entity that is shown in Figure 11.

For reasons of brevity, the deeper deconstruction of the VM4SEC platform in Level-2 DFDs is not provided in the present paper. In addition to this, detailed sequence diagrams have been derived following the common UML notation for all the main functionalities of the VM4SEC platform. Since this information is too technical, we decided to exclude it from the present report.

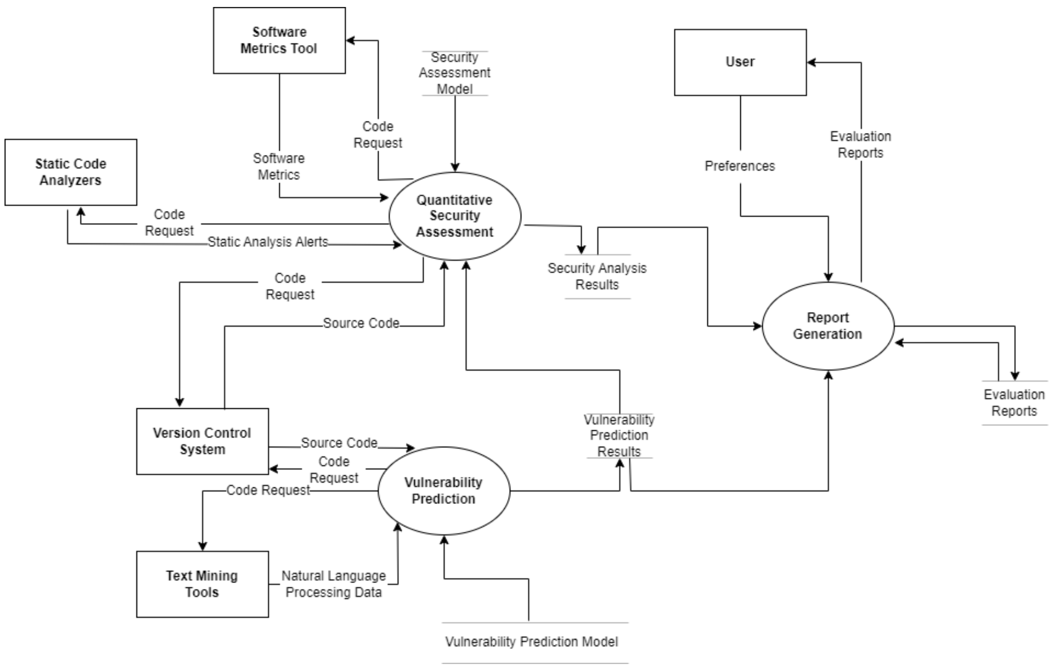


**Figure 11.** The Level-1 data flow diagram (DFD) of the VM4SEC security monitoring platform

The DFDs were highly useful for deriving the component diagrams of the VM4SEC platform, which are necessary for its final development and deployment. A component is an individual functional unit that is integrated into the broader system. Each component has its own interface, which enables its interaction with other components. In contrast to the DFDs, the component diagrams focus on providing a high-level representation of the system, avoiding technical details, and giving emphasis to the interaction between the components of the system. For defining the component diagram, we utilized the common UML notation. The overall component diagram of the VM4SEC Platform is illustrated in Figure 12.

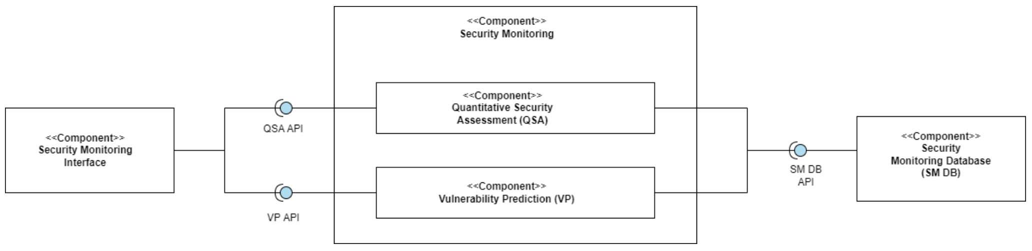

<<Component>> Security Monitoring
<<Component>> Quantitative Security Assessment (QSA)
<<Component>> Vulnerability Prediction (VP)
<<Component>> Security Monitoring Interface
QSA API
VP API
SM DB API
<<Component>> Security Monitoring Database (SM DB)

**Figure 12.** The Component Diagram of the VM4SEC security monitoring platform.

As can be seen in Figure 12, the VM4SEC platform consists of four (4) main components, which have either a provided or a requested Application Programming Interface (API). The two core components of the VM4SEC platform are the quantitative security assessment (QSA) and the vulnerability prediction (VP) components, which constitute the back-end of the platform and represent the two core security monitoring features that it provides (see Sections 3.3 for more details). The component named security monitoring interface corresponds to a graphical user interface (GUI) that will enable the user to use the security assessment and vulnerability prediction components and retrieve a visual representation of their outputs. Finally, the security monitoring database component is responsible for storing the results of the analysis that are produced by the quantitative security assessment and vulnerability prediction components. As will be discussed in Section 3.4, the core components of the VM4SEC platform will be implemented as microservices, which will provide their functionalities through RESTfull APIs. These microservices will be deployed utilizing the Docker Engine.

### *3.3. Configuration of the Core Security Monitoring Mechanisms*

As described in the previous section, the two core elements of the VM4SEC platform will be the *quantitative security assessment* and the *vulnerability prediction models*, as they were found to be the most interesting and appealing novel solutions according to the Company (as reported by the industrial study presented in Section 3.1.1). However, these models need to be adapted to the specific needs of the Company, and particularly they should be appropriately calibrated in order to satisfy the unique characteristics of the software applications on which they will be employed. In the present section, we give a brief description of these two security monitoring mechanisms that we have proposed in the past and we describe how we calibrated these parameters in order to be in line with the needs of the subject Company.

#### 3.3.1. Quantitative Security Assessment

As part of the VM4SEC platform, we will deploy a novel mechanism able to compute high-level measures that reflect important security aspects of the analyzed software based on the results of security-related static analysis. In particular, we will utilize a state-of-the-art hierarchical security assessment model that was proposed by Siavvas et al. [17], which, based on the guidelines of the ISO/IEC 25010 [42] security standard and a set of thresholds and weights systematically aggregates the results of static analysis in order to compute high-level security scores, including the security index, which is a score that reflects the security level of the software product under analysis. The general structure of the model is illustrated in Figure 13.

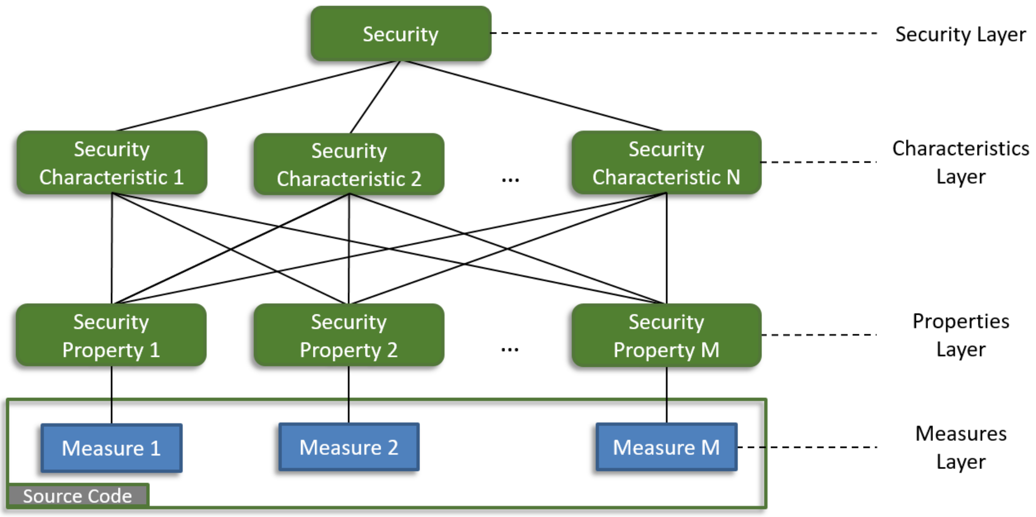


**Figure 13.** The general structure of the selected security assessment model.

As can be seen by inspecting Figure 13, the model hierarchically decomposes the notion of security into a set of security characteristics (e.g., Confidentiality, Integrity, etc.), which are further decomposed into a set of low-level security properties (e.g., Null Pointer, Buffer Overflow, etc.). The security properties correspond to categories of security issues that the software application may exhibit, and are quantified through dedicated metrics, which are, in fact, the densities of static analysis alerts that correspond to these categories. In brief, initially, static analysis is applied to a given software application and the densities of each one of the selected vulnerability categories are computed. Subsequently, based on a set of thresholds, a score between 0 and 1 is assigned to their corresponding security properties, indicating how well the application avoids the associated vulnerability category. Then, based on a set of weights, the scores of the security properties are aggregated in order to compute the scores of the security characteristics of the model. Finally, the overall Security Index of the system is computed by taking the average of the scores of the security characteristics of the model. For more information about the security model, we refer the reader to the original paper [17].

From the above analysis, it is clear that the security characteristics, the security properties, the weights, and the thresholds of the model constitute its main design parameters that need to be defined properly in order to fit the needs of the company. This information was derived from the industrial case study that was presented in Section 3.1.1, and particularly, from the second step of the survey and the fourth block of the focus group. More specifically, it was decided that the final model would have three security characteristics, namely Confidentiality, Availability, and Integrity, as these were recognized as the most important security aspects of the software projects that are developed by the company. In addition to this, based on the information gathered from the participants of the industrial case study, those three characteristics were considered equally important for the overall security of the developed systems, indicating that they should receive equal weight in the model.

The most important vulnerability categories according to the company (which act as the security properties of the model) were found to be: Null Pointer, Weak Cryptography, Insufficient/Incorrect Logging, Security Misconfiguration, and Non-optimum Resource Allocation. The thresholds of the model were calibrated utilizing a popular benchmarking approach [48,49] based on a set of reference software projects that were selected by the software engineers of the company, as representative examples of the software applications that they build for their clients. Finally, the weights of the model were determined based on the popular SMARTS/SMARTER approach [50], by using the responses of the participants in the second step of the survey described in Section 3.1.1 in order to rank the vulnerability categories and identify their relative importance to the selected security characteristics. In fact, the rankings were devised based on the responses of the second step of the survey, which were then presented to the participants during the focus group for final verification and revision. This relative importance was reflected by the SMARTS/SMARTER approach to numerical values, in the form of weights.

### 3.3.2. Vulnerability Prediction

As the second core security monitoring mechanism of the VM4SEC platform, we will deploy a novel mechanism that enables the early identification of security vulnerabilities in software systems. In particular, we will utilize the models that were initially examined by Kalouptsoglou et al. [51] and were extended in [52]. These deep learning models are based on text mining methods and use textual software attributes extracted from the source code in order to predict which software components of an analyzed application are most likely to contain vulnerabilities. An overview of the utilized vulnerability prediction model is illustrated in Figure 14.

As can be seen in Figure 14, the source code of the analyzed software component is provided as input to the system. First, the tool applies text mining to extract the software attributes of the software component and then it tokenizes it to produce sequences of tokens. Then the token sequences are encoded in numerical vectors, which are called embedding vectors, through a sophisticated external algorithm entitled word2vec [53]. Subsequently, the embedding vectors are passed to the embedding layer of the neural network and then to the hidden layers of a convolutional neural network (CNN). Through hyperparameter tuning, the optimal prediction model is generated. Hence, when a new software component is analyzed, the vulnerability predictor can classify it as vulnerable or not based on what it has learned.

In order to construct a vulnerability prediction model that could satisfy the Company's security needs (as reported by the industrial study presented in Section 3.1.1), we created, with the collaboration of a group of Company's developers, a vulnerability dataset and then we trained and validated our model on this dataset. For the dataset construction, we followed the approach of searching those commits that fix a vulnerability in Company's projects [54–56] that are stored in GitHub repositories. To be precise, Company's GitHub projects were collected and then the history of commits to each of them was searched. The search was based on finding certain combinations of keywords within the message accompanying each commit, e.g., "vulnerability repair", "dos repair", "dos prevention", "exploit prevention", "fix CWE", etc. In Figure 15, we provide an overview of the implemented tool.

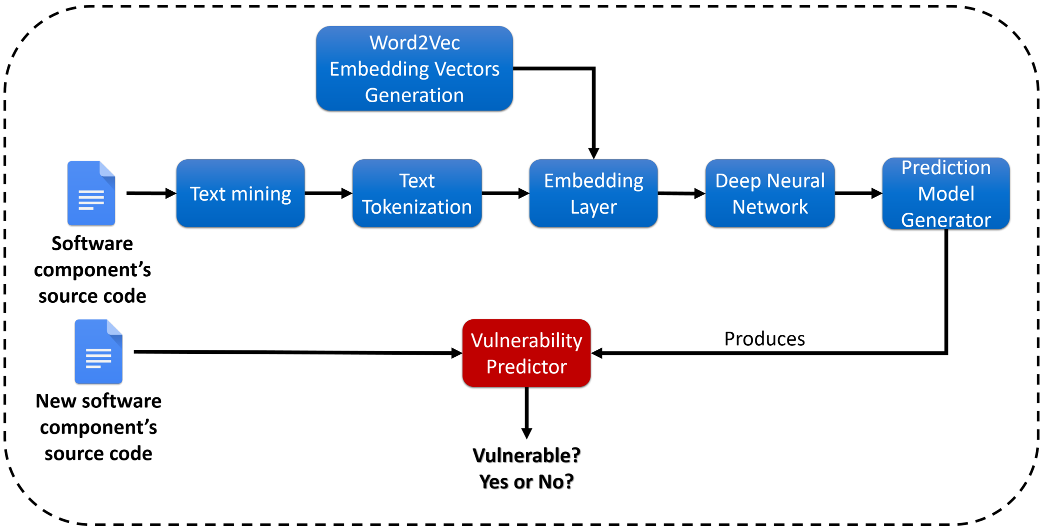
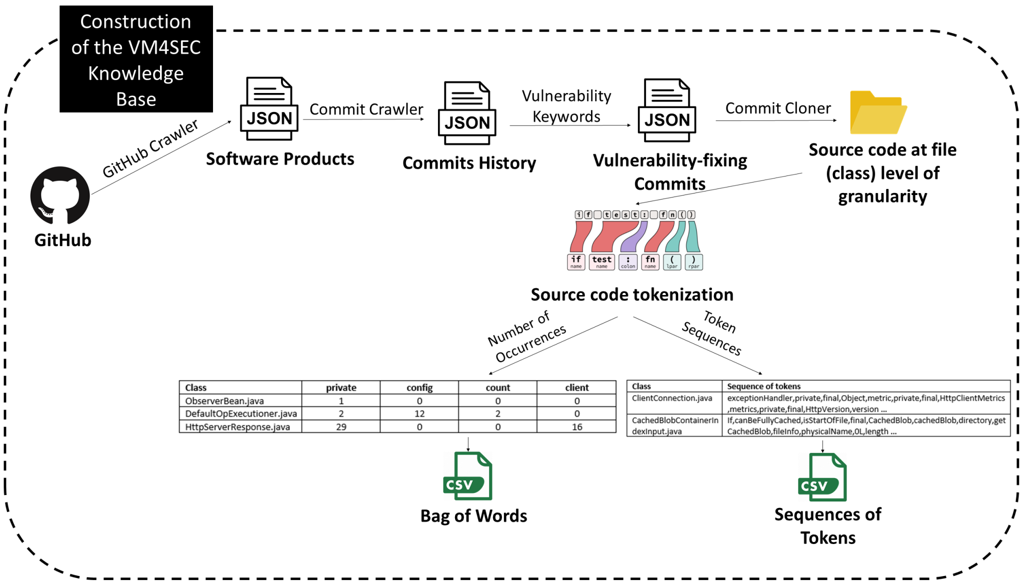

Word2Vec Embedding Vectors Generation
Software component's source code
Text mining
Text Tokenization
Embedding Layer
Deep Neural Network
Prediction Model Generator
Produces
New software component's source code
Vulnerability Predictor
Vulnerable? Yes or No?

**Figure 14.** An overview of the text-mining-based VM4SEC vulnerability prediction model.



| Class | private | config | count | client |
|---|---|---|---|---|
| ObserverBean.java | 1 | 0 | 0 | 0 |
| DefaultOpExecutioner.java | 2 | 12 | 2 | 0 |
| HttpServerResponse.java | 29 | 0 | 0 | 16 |

| Class | Sequence of tokens |
|---|---|
| ClientConnection.java | exceptionHandler,private,final,Object,metric,private,final,HttpClientMetrics ,metrics,private,final,HttpVersion,version ... |
| CachedBlobContainerIndexInput.java | If,canBeFullyCached,isStartOfFile,final,CachedBlob,cachedBlob,directory,getCachedBlob,fileInfo,physicalName,0L,length ... |

**Figure 15.** Overview of the construction of the VM4SEC knowledge base.

As can be seen in Figure 15, the source of the dataset is the GitHub software repository. The tool consists of three main functionalities:

- **GitHubCrawler**: Collects software projects from GitHub for a selected programming language (i.e., Java in our case).
- **CommitCrawler**: Receives the list of software projects and obtains for each project the history of commits. Each commit is accompanied by a message where the developer states the reason for making the particular commit. CommitCrawler searches these messages for specific predefined combinations of keywords indicating changes in order to fix vulnerabilities.
- **CommitCloner**: Downloads the source code of the parent version of the files that underwent changes in each selected commit, and downloads also the latest version of the software projects (where no vulnerability has been found yet).

These two sets of Java files that are downloaded by the *CommitCloner* constitute the VM4SEC Vulnerability Knowledge Base. The tokenized code of these Java files can be formatted into two different ways: (1) bag of words and (2) sequences of tokens. In the former, code is represented as a set of words/tokens along with their number of occurrences in the code, whereas in the latter, code is represented as sequences of words/tokens in the same order that they appear in the actual source code.

During the developing of our vulnerability prediction model, we employed and compared both source code representation methods: Bag of Words (BoW) and sequences of tokens. We used these two forms of the constructed dataset to train and evaluate machine learning (ML) models capable of identifying vulnerabilities in the source code. As depicted in Figure 15, BoW is a numerical representation of each source code file putting particular emphasis on the existing tokens and the frequency in which they appear in the code. In particular, in the BoW method each source code file is represented by a numerical vector, the cells of which correspond to the frequency of each token (i.e., instruction/keyword), i.e., the number of its occurrences in the corresponding source code file. On the other hand, sequences of tokens is a way of representing software components (e.g., files) as sequences of the words that exist in the specific component, taking into consideration the words' position in each sequence. In this case, for the numerical representation of the tokens, word embedding techniques are applied. In particular, we examined the capacity of the word2vec [53] and fastText algorithms [57], and we compared each by conducting an empirical evaluation.

More specifically, word2vec is a ML model proposed by Mikolov et al. [53] that can learn word embedding vectors by predicting the context of the words within a given text corpus. The generated vectors, which are numerical representations of the words, are positioned in such a way that similar words are closer together in the vector space. This way, word embedding representations enhance the attempt of the text-mining-based models to capture syntactic and semantic patterns through the text (i.e., through the source code in VP case). An extension of word2vec is fastText, proposed by Bojanowski et al. [57]. It introduces sub-word information into the word embeddings. In other words, it considers tokens as smaller sub-word units and produces embedding vectors for them. This way, in contrast with word2vec, fastText can handle out-of-vocabulary words, since it can represent unseen words based on the embeddings of the sub-words that reside in these unknown words.

Regarding the ML models that we employed for the construction of our VPM, we focused on deep learning architectures such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) units which is an RNN variant, and Convolutional Neural Networks (CNNs). The aforementioned architectures are able to receive and learn sequential data as the sequences of the tokens that exist in the source code. In particular, LSTM is specifically designed for learning sequential data, capturing long-term dependencies by employing memory units and gates [58]. CNNs are commonly utilized in image processing tasks, but one-dimension (1-D) convolutional layers can be easily utilized for learning sequences. To find the optimal model of each examined deep learning architecture, during the prediction model generator phase (see Figure 14), we applied hyperparameter tuning using a grid search process [59]. To develop the models, we utilized the Keras (https://keras.io/api/models/sequential/, accessed on 27 April 2023) library of the TensorFlow framework (https://www.tensorflow.org/, accessed on 27 April 2023). The training and evaluation processes took place on the CUDA platform (https://developer.nvidia.com/cuda-toolkit, accessed on 27 April 2023) installed on an NVIDIA 3060 RTX GPU.

For the empirical evaluation of the aforementioned models and techniques, we employed an evaluation scheme based on the process of cross-validation. The k-fold cross-validation process is a method of dividing the whole training dataset into k folds recursively considering, each time, one different fold as the testing one. We compute the predictive accuracy of the models for k folds, and after the process is completed, we compute the average accuracy. This way, we avoid putting data bias on the model. For the evaluation of the efficiency of the models, we do not use only the accuracy metric but also precision, recall, $F_1$-score, and $F_2$-score in order to obtain a complete estimation of the predictive power of the produced models, considering True Positives (TPs), True Negatives (TNs), False Positives (FPs), and False Negatives (FNs). The formulas of the evaluation metrics, which were computed using the scikit-learn (https://scikit-learn.org/stable/modules/model_evaluation.html, accessed on 27 April 2023) python library, are provided in the equations below:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \tag{4}$$

$$F_2 = \frac{5 \times precision \times recall}{4 \times precision + recall} \tag{5}$$

The architecture of the best model is presented in Table 9. Table 10 contains the values of the utilized evaluation metrics of our best model, which is based on the combination of CNN and word2vec algorithms. For reasons of completeness, we provide also the results of the BoW method, which uses an ensemble ML algorithm called Random Forest [60]. As can be seen by Table 10, the representation method of sequences of tokens seems to be superior from the BoW method. Although BoW achieves a higher precision, sequences of tokens succeed higher scores in all the other evaluation metrics. Hence, the sequences of tokens method, which uses CNN and word2vec, proved to be much more efficient in identifying vulnerabilities, even if it produces slightly more FPs. In fact, the $F_2$-score, which is considered more important for the case of vulnerability prediction as it considers both Recall and Precision, but puts more emphasis on Recall, of the CNN with word2vec model was found to be much higher compared to the score of the model that utilized the BoW approach.

**Table 9.** The selected hyperparameters of the CNN model.

| **Hyperparameter Name** | **Value** |
|---|---|
| Number of Layers | 3 (Embedding–Convolutional–Dense) |
| Number of Convolutional Layers | 1 |
| Embedding Size | 300 |
| Number of Filters | 128 |
| Kernel Size | 5 |
| Pooling | global max pooling |
| Weight Initialization Technique | glorot uniform (Xavier) |
| Learning Rate | 0.01 |
| Gradient Descent Optimizer | Adam |
| Batch Size | 64 |
| Activation Function | relu |
| Output Activation Function | sigmoid |
| Loss Function | binary cross-entropy |

**Table 10.** The evaluation metrics of the vulnerability prediction model.

| Evaluation Metric | Sequences | BoW |
|---|---|---|
| Accuracy | 85.87 | 83.06 |
| Precision | 81.20 | 85.70 |
| Recall | 91.25 | 79.50 |
| $F_1$-score | 85.83 | 82.43 |
| $F_2$-score | 88.98 | 80.63 |

## *3.4. Implementation and Deployment of the VM4SEC Platform*

As depicted in Figure 1, the final step of our process is the actual implementation and deployment of the VM4SEC security monitoring platform. We decided to build the VM4SEC platform as a cloud-based web application. The reasoning behind this choice was the fact that the Cloud provides (i) increased visibility, (ii) high accessibility (via the Internet), and (iii) ease of use (since the tedious part of installing the application is avoided). In addition to this, the final application is independent of the final platform and operating system on which the end users work, thus providing a cross-platform experience that would be difficult to achieve in the case of offline applications.

Among the various architectural patterns for building cloud-based applications [46,61,62], we decided to use the *Microservice Architecture (MOA)* pattern [63] over the traditional Service-oriented Architecture (SOA) pattern. Both patterns are based on the concept of implementing the main functionalities of the application as individual services. The main difference between the two patterns is that while SOA requires all the services to be centrally implemented in the form of a monolithic application, in MOA, similar functionalities are grouped into components (i.e., microservices) with their own lifecycles, which can then be distributed over a network. These services can then collaborate together in order to form a broader application. Some of the main advantages of MOA over SOA (which helped us reach our final decision) are listed below [63]:

- Microservices can be deployed independently;
- Microservices can be implemented using different technologies;
- Microservices can be developed quickly, and deployed and maintained by a small, independent team;
- Microservices offer modular maintenance.

A high-level view of the VM4SEC platform following the MOA pattern is provided in Figure 16. As can be seen in Figure 16, each one of the main modules of the overall platform, namely the quantitative security assessment and the vulnerability prediction modules, is implemented as an individual microservice. These microservices expose their functionalities through RESTfull APIs, in order to facilitate their utilization by third-party applications. A web user interface (UI) is also provided in order to facilitate the utilization of the security monitoring services by non-technical stakeholders and provide inference through proper visualization of their results. The overview presented in Figure 16 is in line with the component diagram presented in Figure 12.

For the actual deployment of the microservices, the Docker Engine was used. Docker is an Enterprise Container Platform that allows applications to be packaged as individual containers along with their required parts (e.g., tools, configuration, dependencies, etc.) and communicate with each other through dedicated channels. In fact, the microservices of the VM4SEC Platform are implemented as individual Docker Images, which are then deployed as independent Docker Containers.

Docker encompasses a set of key features that played an important role in our final selection. First of all, each microservice (i.e., Docker Image) can be developed independently of one another, adopting different tools, frameworks, configurations, etc., without posing any restrictions to the other microservices. Hence, Docker provides flexibility in the sense that it allows individual applications to be implemented using highly different languages and technologies, encouraging in this way agile software development. In addition, the implemented Docker Images can be deployed and executed as Docker Containers either on the same machine or distributed over different locations. The execution of one Docker Container does not affect the execution of the others, increasing, in that way, the reliability of the overall platform. Finally, the installation of the final integrated toolbox becomes an easy process, as it is reduced to the deployment of the individual Docker Containers that correspond to the microservices of the platform, which can be performed automatically based on a description provided as a compose file.

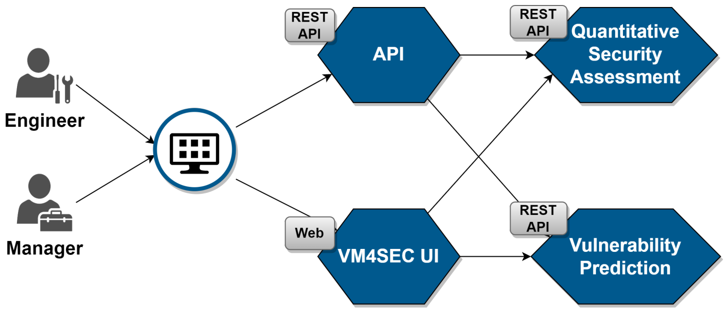


**Figure 16.** The overview of the VM4SEC security monitoring platform utilizing the Microservice-oriented Architectural (MOA) pattern.

The final deployment diagram of the VM4SEC security monitoring platform is depicted in Figure 17. As can be seen, the client–server approach is adopted, and therefore the VM4SEC platform consists of an Application Server, which corresponds to the back-end of the system, and a Client, which corresponds to the front-end (i.e., user interface) of the system. With respect to the back-end, as already stated, the two core security monitoring mechanisms are implemented as individual services, which are deployed as independent Docker containers. In each container, a dedicated lightweight database (i.e., a MongoDB) is provided for storing the results of the analysis performed by each service. Those Docker containers are deployed on a Docker Engine environment, which acts as an orchestrator of the services, mapping the incoming requests to the corresponding service. The Client (i.e., front-end) of the VM4SEC platform, since it is a cloud-based web application, is a web page that can be accessed through an Internet Browser (e.g., Google Chrome, Mozilla Firefox, etc.). The implementation of the Web UI was implemented utilizing cutting-edge technologies, in order to be up-to-date and ensure its longevity. In particular, it has been implemented in JavaScript, utilizing the React.js framework, which is one of the most popular frameworks for building cloud-based applications, in conjunction with MD- Bootstrap, ASP.NET Core, and PostgreSQL5.

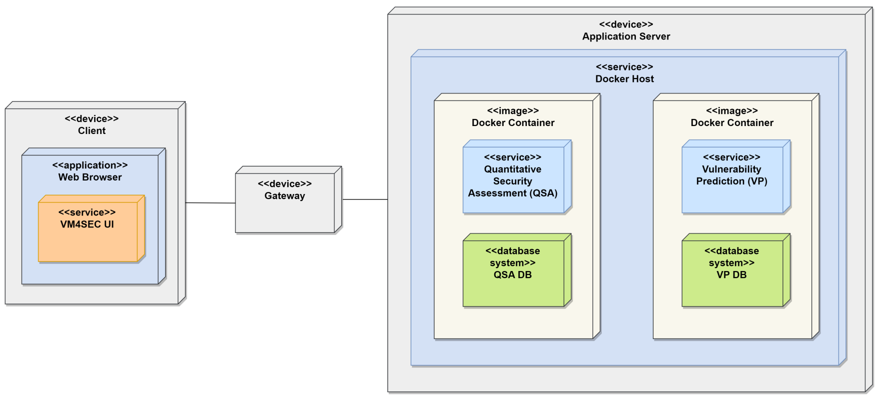


**Figure 17.** The Deployment Diagram of the VM4SEC security monitoring platform.

## 4. Conclusions and Future Work

In the present paper, we demonstrated how novel security monitoring mechanisms can be turned into a practical solution properly configured to satisfy custom security needs, through an industrial case study based on a real software development company. Initially, we conducted an empirical study in order to collect information from the subject company about (i) the current state of their security monitoring activities, (ii) their actual needs for security monitoring during the development process, and (iii) the most suitable security monitoring techniques to be deployed into their pipelines along with the appropriate configuration. This was achieved through a 2-step survey and a focus group conducted with 16 participants from the subject company. Subsequently, we elicited the requirements and defined the main use cases of the envisaged platform, and based on this information we designed its overall architecture. This led to the implementation of the VM4SEC platform, a security monitoring platform that provides cutting-edge security monitoring mechanisms, tailored to the needs of the company that was used as the subject of our case study. The present paper can be used as an example of how similar novel security monitoring solutions can be made operational in the form of practical tools that are properly configured in order to satisfy the needs of a given company.

To the best of our knowledge, the VM4SEC platform is the only static-analysis-based platform that is available in the literature that provides quantitative security assessment of software products based on cutting-edge security assessment models. It is also the first platform that provides fully operational text-mining-based VPMs, which can be used directly for identifying potentially vulnerable components in software products under development. Finally, an important contribution of the present work is the presentation of a formal approach for building similar security monitoring platforms, i.e., for configuring and adapting novel security monitoring solutions to meet the needs of a specific company and/or application domain, and for integrating them into a unified monitoring platform.

Several directions for future work can be identified. First of all, after building the VM4SEC platform, we are planning to evaluate its usefulness and practicality through its pilot usage by the subject company. More specifically, the VM4SEC platform will be deployed on the premises of Onelity (i.e., the subject company of the present study) and will be utilized as part of their SDLC to monitor the security level of actual software products that are being developed. After a sufficient period of hands-on experience with the platform, a qualitative evaluation will be conducted, by retrieving feedback from its actual users with respect to its usability, usefulness, and practicality, as well as with respect to how well it reflects their original needs, as expressed during the survey and focus group that were conducted at the beginning of the project and reported in the present paper.

Secondly, the potential integration of additional features will be also considered, based on the active feedback that we will receive from its actual users, in order to further improve the usability and practicality of the platform. More specifically, from a technical viewpoint we are planning to extend the VM4SEC platform, by integrating additional static code analyzers such as SonarQube, Coverity, and Fortify. This will allow the platform to detect additional types of security issues and compute new security metrics. Finally, with respect to vulnerability prediction, we are planning to integrate text mining VPMs that are based on pre-trained large language models such as GPT and BERT, since our latest work showcased their potential in being used as the basis for vulnerability prediction [29].

**Author Contributions:** Conceptualization: M.S., D.T. (Dimitrios Tsoukalas), and I.K.; methodology: M.S., I.K. and D.T. (Dimitrios Tsoukalas); software: M.S., I.K. and D.T. (Dimitrios Tsoukalas); validation: M.S. and D.T. (Dimitrios Tsoukalas); formal analysis: I.K., D.T. (Dimitrios Tsoukalas), M.S., E.M., G.M., D.K. and D.T. (Dimitrios Tzovaras); investigation: I.K. and D.T. (Dimitrios Tsoukalas); resources: D.K.; related work: E.M. and D.K.; data curation: I.K. and D.T. (Dimitrios Tsoukalas); writing—original draft preparation: M.S., I.K. and D.T. (Dimitrios Tsoukalas); writing—review and editing: M.S., D.T. (Dimitrios Tsoukalas), I.K.; visualization: I.K., D.T. (Dimitrios Tsoukalas) and M.S.; supervision: D.K., G.M. and D.T. (Dimitrios Tzovaras); project administration: D.K.; funding acquisition: D.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was carried out as part of the project VM4SEC (Project code: KMP6-0083702) under the framework of the Action Investment Plans of Innovation of the Operational Program

Central Macedonia 2014 2020, that is co-funded by the European Regional Development Fund and Greece.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Informed consent was obtained from all subjects involved in the study.

**Data Availability Statement:** The data related to the present work are available upon request.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

| | |
|---|---|
| API | Application Programming Interface |
| ASA | Automatic Static Analysis |
| AST | Abstract Syntax Tree |
| BERT | Bidirectional Encoder Representations from Transformers |
| BoW | Bag of Words |
| CFG | Control-flow Graph |
| CNN | Convolutional Neural Network |
| CPG | Code Property Graph |
| CWE | Common Weakness Enumeration |
| DFD | Data Flow Diagram |
| DFG | Data-Flow Graph |
| FN | False Negative |
| FP | False Positive |
| FR | Functional Requirement |
| GPT | Generative Pre-trained Transformer |
| GUI | Graphical User Interface |
| IT | Information Technology |
| LSTM | Long Short-Term Memory |
| MCDM | Multi-Criteria Decision-Making |
| ML | Machine Learning |
| MOA | Microservice-Oriented Architecture |
| QA | Quality Assurance |
| QSA | Quantitative Security Assessment |
| REST | REpresentational State Transfer |
| RNN | Recurrent Neural Network |
| SaaS | Software-as-a-Service |
| SDLC | Software Development Lifecycle |
| SOA | Service-Oriented Architecture |
| SSRF | Server-Side Request Forgery |
| TN | True Negative |
| TP | True Positive |
| UI | User Interface |
| UML | Unified Modeling Language |
| VPM | Vulnerability Prediction Model |
| VP | Vulnerability Prediction |
| XSS | Cross-Site Scripting |

## References

1. McGraw, G. *Software Security: Building Security In*; Addison-Wesley Professional: Boston, MA, USA, 2006.
2. Williams, L. Secure Software Lifecycle Knowledge Area Version 1.0.2. *CyBok* **2021**.
3. Luszcz, J. Apache Struts 2: How technical and development gaps caused the Equifax Breach. *Netw. Secur.* **2018**, *2018*, 5–8. [CrossRef]
4. Carvalho, M.; DeMott, J.; Ford, R.; Wheeler, D.A. Heartbleed 101. *IEEE Secur. Priv.* **2014**, *12*, 63–67. [CrossRef]
5. Prevezianou, M.F. WannaCry as a Creeping Crisis. In *Understanding the Creeping Crisis*; Palgrave Macmillan: Cham, Switzerland, 2021; pp. 37–50. [CrossRef]
6. Sopariwala, S.; Fallon, E.; Asghar, M.N. Log4jPot: Effective Log4Shell Vulnerability Detection System. In Proceedings of the 2022 33rd Irish Signals and Systems Conference (ISSC), Cork, Ireland, 9–10 June 2022; pp. 1–5. [CrossRef]

7. Chess, B.; McGraw, G. Static analysis for security. *Secur. Priv. IEEE* **2004**, *2*, 76–79. [CrossRef]
8. Howard, M. *Writing Secure Code*; Microsoft Press: Redmond, WA, USA, 2003.
9. Wurster, G.; van Oorschot, P.C. The developer is the enemy. In Proceedings of the NSPW '08: Proceedings of the 2008 Workshop on New Security Paradigms, Twente, The Netherlands, 8–11 September 2008; pp. 89–97. [CrossRef]
10. Green, M.; Smith, M. Developers are Not the Enemy!: The Need for Usable Security APIs. *IEEE Secur. Priv.* **2016**, *14*, 40–46. [CrossRef]
11. Howard, M.; LeBlanc, D.; Viega, J. *24 Deadly Sins of Software Security*; McGraw-Hill: New York, NY, USA, 2010; p. 443.
12. Bholanath, R. Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software. In Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita, Japan, 14–18 March 2016; Volume 1, pp. 470–481. [CrossRef]
13. Morrison, P.; Moye, D.; Pandita, R.; Williams, L. Mapping the Field of Software Security Metrics. *Inf. Softw. Technol.* **2014**, *102*, 146–159.
14. Alshammari, B.; Fidge, C.; Corney, D. A hierarchical security assessment model for object-oriented programs. In Proceedings of the 2011 11th International Conference on Quality Software, Madrid, Spain, 13–14 July 2011; pp. 218–227. [CrossRef]
15. Xu, H.; Heijmans, J.; Visser, J. A practical model for rating software security. In Proceedings of the 2013 IEEE Seventh International Conference on Software Security and Reliability Companion, Madrid, Spain, 13–14 July 2013; pp. 231–232. [CrossRef]
16. Zafar, S.; Mehboob, M.; Naveed, A.; Malik, B. Security quality model: An extension of Dromey's model. *Softw. Qual. J.* **2015**, *23*, 29–54. [CrossRef]
17. Siavvas, M.; Kehagias, D.; Tzovaras, D.; Gelenbe, E. A hierarchical model for quantifying software security based on static analysis alerts and software metrics. *Softw. Qual. J.* **2021**, *29*, 431–507. [CrossRef]
18. Medeiros, N.; Ivaki, N.; Costa, P.; Vieira, M. Trustworthiness models to categorize and prioritize code for security improvement. *J. Syst. Softw.* **2023**, *198*, 111621. [CrossRef]
19. Zagane, M.; Abdi, M.K.; Alenezi, M. Deep learning for software vulnerabilities detection using code metrics. *IEEE Access* **2020**, *8*, 74562–74570. [CrossRef]
20. Pakshad, P.; Shameli-Sendi, A.; Khalaji Emamzadeh Abbasi, B. A security vulnerability predictor based on source code metrics. *J. Comput. Virol. Hacking Tech.* **2023**, 1–19. [CrossRef]
21. Bassi, D.; Singh, H. The Effect of Dual Hyperparameter Optimization on Software Vulnerability Prediction Models. *e-Inform. Softw. Eng. J.* **2023**, *17*, 230102. [CrossRef]
22. Hovsepyan, A.; Scandariato, R.; Joosen, W.; Walden, J. Software vulnerability prediction using text analysis techniques. In Proceedings of the 4th International Workshop on Security Measurements and Metrics, Lund, Sweden, 21 September 2012; pp. 7–10. [CrossRef]
23. Li, Z.; Zou, D.; Xu, S.; Ou, X.; Jin, H.; Wang, S.; Deng, Z.; Zhong, Y. Vuldeepecker: A deep learning-based system for vulnerability detection. *arXiv* **2018**, arXiv:1801.01681. [CrossRef]
24. Zhou, Y.; Liu, S.; Siow, J.; Du, X.; Liu, Y. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Adv. Neural Inf. Process. Syst.* **2019**, *32*. [CrossRef]
25. Tang, W.; Tang, M.; Ban, M.; Zhao, Z.; Feng, M. CSGVD: A deep learning approach combining sequence and graph embedding for source code vulnerability detection. *J. Syst. Softw.* **2023**, *199*, 111623. [CrossRef]
26. Fu, M.; Tantithamthavorn, C. LineVul: A transformer-based line-level vulnerability prediction. In Proceedings of the 19th International Conference on Mining Software Repositories, Pittsburgh, PA, USA, 23–24 May 2022; pp. 608–620. [CrossRef]
27. Hanifi, K.; Fouladi, R.F.; Unsalver, B.G.; Karadag, G. Software Vulnerability Prediction Knowledge Transferring Between Programming Languages. *arXiv* **2023**, arXiv:2303.06177. [CrossRef]
28. Chen, Y.; Ding, Z.; Chen, X.; Wagner, D. DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection. *arXiv* **2023**, arXiv:2304.00409. [CrossRef]
29. Kalouptsoglou, I.; Siavvas, M.; Ampatzoglou, A.; Kehagias, D.; Chatzigeorgiou, A. An empirical comparison of Transformer-based models in Vulnerability Prediction. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2023.
30. Lenarduzzi, V.; Lujan, S.; Saarimaki, N.; Palomba, F. A critical comparison on six static analysis tools: Detection, agreement, and precision. *arXiv* **2021**, arXiv:2101.08832. [CrossRef]
31. Smith, J.; Do, L.N.; Murphy-Hill, E. Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security. In Proceedings of the Sixteenth Symposium on Usable Privacy and Security, Berkeley, CA, USA, 10–11 August 2020.
32. Faisal, F.; Elshoush, H.T. Input Validation Vulnerabilities in Web Applications: Systematic Review, Classification, and Analysis of the Current State-of-the-Art. *IEEE Access* **2023**, *11*, 40128–40161. [CrossRef]
33. Arzt, S.; Rasthofer, S.; Fritz, C.; Bodden, E.; Bartel, A.; Klein, J.; Le Traon, Y.; Octeau, D.; McDaniel, P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM Sigplan Not.* **2014**, *49*, 259–269. [CrossRef]
34. Johnson, B.; Song, Y.; Murphy-Hill, E.; Bowdidge, R. Why don't software developers use static analysis tools to find bugs? In Proceedings of the 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18–26 May 2013; pp. 672–681. [CrossRef]
35. Trautsch, A.; Herbold, S.; Grabowski, J. Are automated static analysis tools worth it? An investigation into relative warning density and external software quality. *arXiv* **2021**, arXiv:2111.09188. [CrossRef]

36. Ge, X.; Fang, C.; Bai, T.; Liu, J.; Zhao, Z. An Empirical Study of Class Rebalancing Methods for Actionable Warning Identification. *IEEE Trans. Reliab.* **2023**, 1–15. [CrossRef]
37. Siavvas, M.; Gelenbe, E.; Kehagias, D.; Tzovaras, D. Static analysis-based approaches for secure software development. In *Security in Computer and Information Sciences: First International ISCIS Security Workshop 2018, Euro-CYBERSEC 2018, London, UK, 26–27 February 2018, Revised Selected Papers 1*; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 142–157. [CrossRef]
38. Apolinário, V.A.; Bianco, G.D.; Duarte, D.; Leithardt, V.R.Q. Exploring Feature Extraction to Vulnerability Prediction Problem. In *New Trends in Disruptive Technologies, Tech Ethics and Artificial Intelligence: The DITTET 2022 Collection*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 79–90. [CrossRef]
39. Morais, R.; Crocker, P.; Leithardt, V. Nero: A Deterministic Leaderless Consensus Algorithm for DAG-Based Cryptocurrencies. *Algorithms* **2023**, *16*, 38. [CrossRef]
40. Seaman, C.B. Qualitative methods in empirical studies of software engineering. *IEEE Trans. Softw. Eng.* **1999**, *25*, 557–572. [CrossRef]
41. Elo, S.; Kyngäs, H. The qualitative content analysis process. *J. Adv. Nurs.* **2008**, *62*, 107–115. [CrossRef]
42. ISO/IEC. *ISO/IEC 25010*; Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models; ISO/IEC: Geneva, Switzerland, 2011.
43. Kitchenham, B.; Pfleeger, S. Principles of survey research part 1: Turning lemons into lemonade. *ACM Sigsoft Softw. Eng. Notes* **2001**, *26*, 16–18. [CrossRef]
44. Clegg, D.; Barker, R. *Case Method Fast-Track: A RAD Approach*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1994.
45. Cockburn, A. *Writing Effective Use Cases*; Pearson Education India: Delhi, India, 2001.
46. Buschmann, F.; Henney, K.; Schmidt, D.C. *Pattern-Oriented Software Architecture, on Patterns and Pattern Languages*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
47. Avison, D.; Fitzgerald, G. *Information Systems Development: Methodologies, Techniques and Tools*; McGraw-Hill: New York, NY, USA, 2003.
48. Vale, G.; Fernandes, E.; Figueiredo, E. On the proposal and evaluation of a benchmark-based threshold derivation method. *Softw. Qual. J.* **2019**, *27*, 275–306. [CrossRef]
49. Baggen, R.; Correia, J.P.; Schill, K.; Visser, J. Standardized code quality benchmarking for improving software maintainability. *Softw. Qual. J.* **2012**, *20*, 287–307. [CrossRef]
50. Edwards, W.; Barron, F.H. SMARTS and SMARTER: Improved simple methods for multiattribute utility measurement. *Organ. Behav. Hum. Decis. Process.* **1994**, *60*, 306–325. [CrossRef]
51. Kalouptsoglou, I.; Siavvas, M.; Tsoukalas, D.; Kehagias, D. Cross-project vulnerability prediction based on software metrics and deep learning. In *International Conference on Computational Science and Its Applications, Proceedings of the Computational Science and Its Applications—ICCSA 2020, Cagliari, Italy, 1–4 July 2020*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 877–893. [CrossRef]
52. Kalouptsoglou, I.; Siavvas, M.; Kehagias, D.; Chatzigeorgiou, A.; Ampatzoglou, A. An empirical evaluation of the usefulness of word embedding techniques in deep learning-based vulnerability prediction. In Proceedings of the EuroCybersec 2021: Security in Computer and Information Sciences, Nice, France, 25–26 October 2021; p. 23. [CrossRef]
53. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781. [CrossRef]
54. Wang, H.; Ye, G.; Tang, Z.; Tan, S.H.; Huang, S.; Fang, D.; Feng, Y.; Bian, L.; Wang, Z. Combining graph-based learning with automated data collection for code vulnerability detection. *IEEE Trans. Inf. Forensics Secur.* **2020**, *16*, 1943–1958. [CrossRef]
55. Bagheri, A.; Hegedűs, P. A comparison of different source code representation methods for vulnerability prediction in Python. In *International Conference on the Quality of Information and Communications Technology, Quality of Information and Communications Technology*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 267–281. [CrossRef]
56. Wartschinski, L.; Noller, Y.; Vogel, T.; Kehrer, T.; Grunske, L. VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python. *Inf. Softw. Technol.* **2022**, *144*, 106809. [CrossRef]
57. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching word vectors with subword information. *arXiv* **2016**, arXiv:1607.04606. https://doi.org/10.48550/arXiv.1607.04606.
58. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
59. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for hyper-parameter optimization. *Adv. Neural Inf. Process. Syst.* **2011**, *24*.
60. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
61. Bass, L.; Clements, P.; Kazman, R. *Software Architecture in Practice: Software Architect Practice_c3*; Addison-Wesley: Boston, MA, USA, 2012.
62. Richards, M. *Software Architecture Patterns*; O'Reilly Media: Sebastopol, CA, USA, 2015; Volume 4.
63. Wolff, E. *Microservices: Flexible Software Architecture*; Addison-Wesley Professional: Boston, MA, USA, 2016.