The IoTAC Software Security-by-Design Platform: Concept, Challenges, and Preliminary Overview

Miltiadis Siavvas*, Erol Gelenbe †, Dimitrios Tsoukalas*, Ilias Kalouptsoglou*, Maria Mathioudaki*, Mert Nakip †, Dionysios Kehagias*, Dimitrios Tzovaras*
* Centre for Research and Technology Hellas, Thessaloniki, Greece
†Institute of Theoretical & Applied Informatics, Polish Academy of Sciences, Gliwice, Poland
siavvasm@iti.gr, seg@iitis.pl, tsoukj@iti.gr, iliaskaloup@iti.gr, mariamathi@iti.gr, mnakip@iitis.pl, diok@iti.gr, dimitrios.tzovaras@iti.gr

***Abstract*—Critical everyday activities handled by modern IoT Systems imply that security is of major concern both for the end-users and the industry. Securing the IoT System Architecture is commonly used to strengthen its resilience to malicious attacks. However, the security of software running on the IoT must be considered as well, since the exploitation of its vulnerabilities can infringe the security of the overall system, regardless of how secure its architecture may be. Thus, we present an IoT Software Security-by-Design (SSD) Platform, which provides mechanisms for monitoring and optimizing the security of IoT software applications throughout their development lifecycle, to validate the broader security of the IoT software. This paper describes the proposed SSD platform that leverages security information from all phases of development, using some novel mechanisms that have been implemented, and which can lead to a holistic security evaluation and future security certification.**

Keywords*—*Internet of Things, Software Security, Requirements Engineering, Static Analysis, Vulnerability Prediction

I. Introduction

Modern Internet of Things (IoT) Systems consist of a large number of interconnected and highly diverse devices, such as sensors, actuators, gateways, etc., often accessible and controllable through the Internet. The high interconnectivity and accessibility of modern IoT Systems, along with the criticality of the daily activities that they monitor and control (e.g. smart living, autonomous driving, industrial control, etc.) render their security an aspect of utmost concern, both for the users and the owning enterprises [1].

An effective way of securing an IoT System is by securing its architecture. This can be achieved through conformance to International IoT Security Standards and the deployment of various security countermeasures, such as intelligent attack detection, prevention, and mitigation mechanisms, security gateways, honeypots, etc. Several initiatives have recently focused on extending well-established IoT Architectures (e.g., the ISO/IEC 30141 Reference Architecture [2]) towards strengthening their security (e.g., SerIoT [1]). Apart from a secure IoT Architecture however, the software that is running on the different nodes of an IoT System should also be considered. As per the "security of the weakest link" principle, if the software contains vulnerabilities, the security of the overall system could be compromised regardless of how secure the overall architecture may be. Hence, to ensure a secure IoT System, the security level of the software running on its nodes should be assessed and optimized throughout its development.

To this end, we develop the Software Security-by-design (SSD) Platform, i.e., a novel software security monitoring and optimization platform that provides mechanisms for assessing and improving the security of IoT software applications, throughout their overall Software Development Lifecycle (SDLC). In particular, SSD allows the developers of an IoT software application to (i) ensure the correct definition of the security requirements, (ii) ensure the adherence of the produced IoT software application to the originally defined requirements, (iii) evaluate the security level of the source code the IoT software application, and (iv) provide recommendations for security improvements. In that way, the SSD Platform provides a more holistic software security assessment, as it covers all the phases of the SDLC horizontally.

The purpose of the present paper is to present the overview of the envisaged SSD Platform and describe its main functionalities, i.e., the main novel mechanisms that have been proposed and developed so far. The SSD Platform is one of the main outcomes of the IoTAC Project, an EU Project funded through the Horizon2020 Programme.

In the sequel, Section 2 discusses the related work focusing on the main challenges that we try to address. Section 3 provides an overview of the broader SSD Platform, whereas Section 3 describes the main novel mechanisms that have been developed so far. Finally, Section 5 concludes the paper and discusses directions for future work.

II. Related Work

According to the *Security-by-Design* paradigm, security should be monitored and optimized at all phases of the SDLC, and particularly during the Requirements, Design, Coding, and Testing phases.

During the *Design* and *Requirement* phases, security can be added by ensuring that the security requirements are correctly defined, since a large portion of software vulnerabilities stem from missing, incorrect, or vague security requirements [3]. Although several approaches for specifying, verifying, and validating functional requirements exist [4]–[6], highly limited

978-1-5386-7097-2/18/$31.00

contributions can be found with respect to non-functional requirements, including security requirements. In particular, several templates and dedicated specification languages have been proposed [3], but their adoption in practice is limited as they are highly complex and tedious to apply. Hence, there is a need for a mechanism able to facilitate the specification of security requirements and enable their verification and validation, preferably in a highly automated way.

During the *Coding* and *Testing* phases of the SDLC, several mechanisms have been proposed for detecting security issues, with static analysis being the most promising solution [7]. However, the fact that their results (i.e., static analysis alerts) are in a very raw and low-level form, hinders their practicality, since the encapsulated security information is not easily conveyed to the end-users. Hence, dedicated tools are needed, able to post-process the results of these tools in order to provide more intuitive security information to developers and project managers, assisting them in making strategic decisions.

Two common post-processing mechanisms are: (i) the *security assessment* models, which aggregate the static analysis results to provide high-level quantified security measures that reflect important security aspects, and (ii) the *vulnerability prediction* models, which highlight software components (e.g., classes, functions) that are likely to contain vulnerabilities. However, despite some notable attempts [8], [9], no well-accepted security models exist in the literature (especially for the case of IoT software). In addition, existing vulnerability prediction models [10] have not demonstrated sufficient results, and do not consider system-level information in order to assess the vulnerability status of software components. Finally, no assessment methodology exists that takes into account information from all the phases of the SDLC and provide a more holistic security evaluation.

Within the context of the IoTAC project, we attempt to address the aforementioned challenges by proposing novel security monitoring and optimization mechanisms for the various phases of the SDLC. We also develop the SSD Platform that integrates these individual security monitoring and optimization solutions, and aggregates their results in order to provide a more holistic security evaluation. The broader evaluation results will be issued in the form of a pseudo-certificate, in order to showcase how the SSD Platform can be leveraged for future security certification activities.

III. Overview of the IoTAC Security-by-Design Platform

The SSD Platform is an independent platform that aims towards offering solutions for monitoring and optimizing the security level of software application running on IoT devices, throughout their overall SDLC. In particular, it enables the user to (i) correctly specify security requirements of a given IoT software application in a uniform and concrete manner, (ii) assess the adherence of the specified security requirements, (iii) detect potential vulnerabilities that reside in the source code of the software application or potentially vulnerable components (i.e., security hotspots), and (iv) quantify the security level of IoT software through the provision of high-level security measures. Solutions will be also provided for validating (in fact, certifying) the overall security of an IoT software application by considering (i) security evaluation results from the aforementioned phase-level security monitoring mechanisms, and (iii) its compliance to selected International Security Standards (e.g., ISO/IEC 27001, IEC 62443-4-2, etc.).

The high-level overview of the SSD Platform is illustrated in Figure 1. As can be seen, the proposed platform consists of three main modules, i.e., the *Design and Requirements Module*, the *Security Assurance Module*, and the *Security Certification Module*. The first two modules provide mechanisms for monitoring and optimizing the security of a given application during the Requirements, Design, Coding, and Testing phases of the SDLC. The third module is responsible for validating the overall security of the analyzed software based on the results of the other two modules, and issuing a certificate (in fact, a pseudo-certificate) that reflects its overall security level. To better understand the overall goal of the SSD Platform, its core modules are briefly described in the rest of this section, while a more detailed description of the main already-developed mechanisms is provided in Section IV.

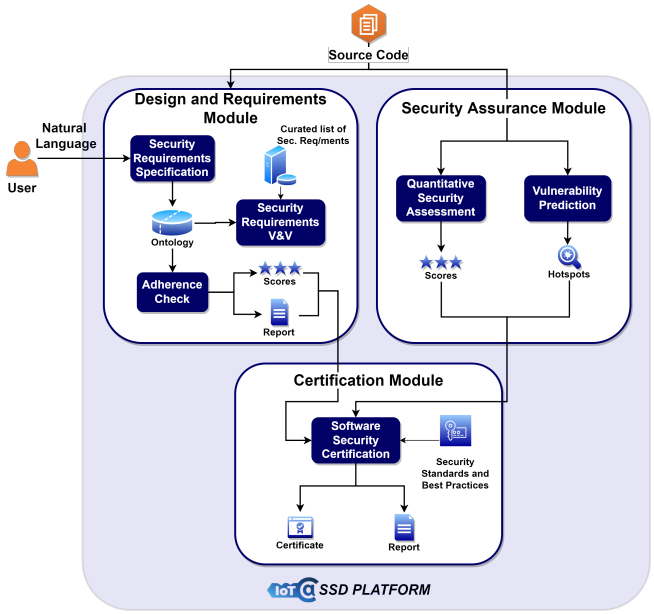

Fig. 1: The high-level overview of the IoTAC Software Security-by-Design (SSD) Platform

The *Design and Requirements Module* is responsible for monitoring and improving the security level of a software application during the Design and Requirement phases of the SDLC. As shown in Figure 1, this module consists of three main mechanisms (i.e., components):

- ***Software Security Requirements Specification***: This mechanism is meant to facilitate the specification of the security requirements of a given software product. It allows the user to define the desired security requirements into natural language (avoiding tedious templates),

processes them to automatically determine their main concepts and underlying semantics, and turns them into a well-structured and unified form.

- ***Software Security Requirements Verification and Validation***: The purpose of this mechanism is to evaluate the correctness and completeness of the software security requirements defined by the user, as well as to provide recommendations regarding their improvement.
- ***Software Security Requirements Adherence Check***: This mechanism is responsible for evaluating whether the final IoT software application adheres to the originally imposed security requirements. More specifically, this mechanism is expected to pinpoint security requirements that either have not been addressed at all, or have partially been implemented and require more technical work.

The *Software Security Assurance Module* is responsible for monitoring and improving the security level of a software application during the Coding and Testing phases of the SDLC. As shown in Figure 1, this module provides the following core mechanisms (i.e., components):

- ***Quantitative Software Security Assessment***: The purpose of this component is to provide quantitative expressions of internal security aspects of an IoT Software application, based on the existence of potential security issues that may reside in their code. This component is based on state-of-the-art concepts from the fields of software quality and software security evaluation (e.g., [8], [9]).
- ***Vulnerability Prediction***: This component is responsible for identifying security hotspots, i.e., software components that are likely to contain vulnerabilities. It is based on vulnerability prediction models that are built based on (i) advanced machine learning techniques (mainly deep learning) and (ii) popular vulnerability datasets.

Finally, the purpose of the *Software Security Certification* module is to provide solutions for validating the overall security of a given IoT software application, focusing on individual security assessments from the various phases of its SDLC that could be used to potentially support its future certification. To do so, this module takes into account (i) project-specific security evaluations (retrieved from the other two modules), and (ii) conformance/compliance to international security standards (e.g., ISO/IEC 27001, IEC-62443-4-2, etc.) and produces a pseudo-certificate reflecting the security level of the analysed IoT software, along with a report on various aspects that require fixing.

IV. Core Elements

This section describes the novel mechanisms that have been developed as part of the SSD Platform, along with details regarding their main elements. To date, novel mechanisms have been developed as part of the *Design and Requirements* module and the *Software Security Assurance* module.

A. Design and Requirements Module

1) Software Security Requirements Specification: The purpose of this component is to aid software engineers formally specify security requirements in a well defined and structured way, without having to utilize tedious formal templates or specification languages. To this end, as part of the SSD Platform, a novel security requirements specification mechanism has been implemented, able to automatically identify the main concepts of a security requirement defined by the user in natural language (i.e., text), and express them in a well-structured and common form [11]. More specifically, the proposed mechanism applies Natural Language Processing techniques (i.e., syntactic and semantic analysis) to identify the main syntactic and semantic concepts of the submitted requirements, and maps these concepts into dedicated Ontology objects (i.e., Actor, Action, etc.), which are stored into a dedicated Ontology. In that way, the submitted raw textual requirements are automatically turned into a formal ontology-based representation. The high-level overview of the proposed mechanism is illustrated in Figure 2.

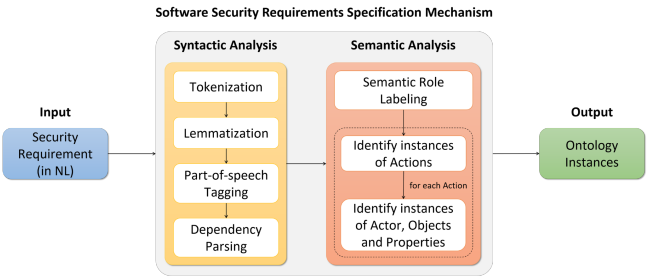

Fig. 2: The high-level overview of the Software Security Requirements Specification mechanism

As can be seen by Figure 2, the proposed mechanism consists of two main steps, namely the *Syntactic Analysis*, which identifies the main grammatical terms (e.g., noun, verb, etc.) of the requirement along with their grammatical relationships (e.g., subject-verb-object, etc.), and the *Semantic Analysis*, which identifies the main semantic concepts (e.g., Action, Actor, Object, etc.) of the requirement along with their semantic relations.

The *Syntactic Analysis* step receives as input the requirement expressed in natural language and applies *tokenization* in order to split the sentence into word tokens, as well as *lemmatization* in order to derive the uninflected form of each word token. Subsequently, it applies *part-of-speech tagging* to identify the grammatical category of each word and *dependency parsing* to determine the grammatical relations between them. For the above procedure, the Mate Tools[1] were employed, since they are widely-used for such tasks.

In the next step, the results of the *Syntactic Analysis*, and particularly the identified grammatical terms and relations, are provided as input to the *Semantic Analysis* mechanism, in order to be mapped to semantic terms and relationships. Initially, a *semantic role labeling* process is performed to assign labels to words or phrases that indicate their semantic concept in the sentence, using the semantic role labeller provided by

[1]https://code.google.com/archive/p/mate-tools/

the Mate tools. However, since this semantic role labeller is able to detect only generic thematical concepts and relations (e.g., acceptor, property, etc.), it has been extended in order to also detect the main requirement-specific concepts (i.e., Actor, Action, etc.), based on a set of custom rules [11]. In brief, as can be seen by Figure 2, initially the Priority and the Action of the requirement are identified. Subsequently, for each Action, the associated Actor and Object are identified. Finally, for each identified Object, several Properties (e.g., requirement prerequisites, etc.) are detected and reported. Finally, all the identified requirement concepts are stored into a dedicated Ontology, called *Security Requirements Knowledge Base*.

2) Software Security Requirements Verification and Validation: As already mentioned, a large portion of software vulnerabilities stem from incorrect, missing, or vague security requirements. The purpose of this component is to verify and validate the correctness and completeness of the user-defined security requirements and provide recommendations for their improvement. To this end, a novel mechanism has been developed, aiming to compare a given security requirement to a curated list of well-defined security requirements (normally retrieved from international standards and other projects), identify inconsistencies, and finally propose refinements [11]. The high-level overview of the aforementioned mechanism is depicted in Figure 3.

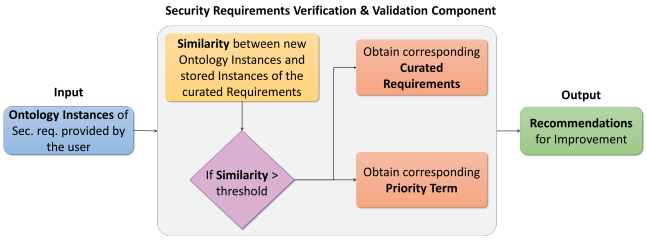

Fig. 3: The high-level overview of the Software Security Requirements Verification and Validation mechanism

As can be seen by Figure 3, the mechanism receives as input the Ontology instances of a security requirement, i.e., its main semantic concepts (e.g., Action, Actor, Object, etc.), as derived by the specification mechanism presented in Section IV-A1. Subsequently, these instances are compared to the ontology instances of the carefully curated list of security requirements that are stored in the *Software Security Requirements Knowledge Base*. In particular, similarity checks are performed between the user-defined requirement and those of the curated list, utilizing popular NLP toolkits (e.g., WordNet with NLTK[2]). Based on the values of the calculated similarity scores, several recommendations for improvement are provided, such as: (i) rephrasing the analyzed security requirement based on a highly similar requirement found in the curated list, (ii) inclusion of additional security requirements that are observed to be closely related to the analyzed requirement, and (iii) changing the priority of the analyzed security requirement based on the priority of similar requirements in the curated list.

[2]https://www.nltk.org/

B. Security Assurance Module

1) Quantitative Security Assessment: The purpose of this component is to provide quantitative security indicators (i.e. security measures) that reflect important security attributes of an IoT application's source code, and help developers identify code-level issues that may correspond to critical vulnerabilities. Those indicators are computed by aggregating security information retrieved from static code-level analysis, which is acknowledged to be one of the most effective mechanisms for detecting vulnerabilities that reside in source code [7], [12].

To this end, as part of the SSD Platform, we developed the ***Security Evaluation Framework (SEF)***, which is a mechanism that evaluates the security level of the source code of a given IoT software application in a quantitative manner, based on the results of security-specific static analysis. More specifically, SEF (i) integrates popular static code analyzers known to detect security issues, and (ii) enables the calculation of high-level security measures by aggregating the low-level results of the security-specific static analysis. High-level security measures are more intuitive and easily understandable even by stakeholders with little or no technical knowledge, such as project managers, facilitating in that way decision making during the overall development of the software application. The high-level overview of SEF is illustrated in Figure 4.

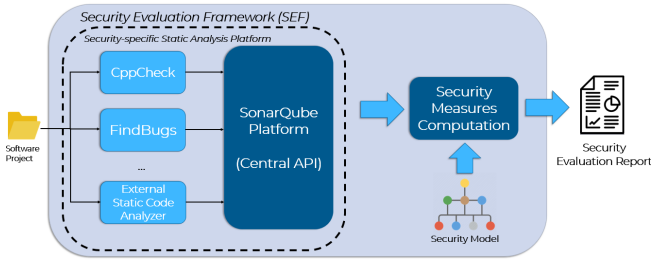

Fig. 4: Overview of the Security Evaluation Framework

As can be seen by Figure 4, SEF receives as input a software application and employs static analysis in order to detect potential security issues (i.e., security-related static analysis alerts) that may reside in the software. This is achieved mainly via a popular static analysis platform, namely SonarQube[3], which is configured in order to detect important security vulnerabilities (e.g., SQL Injection, Cross-site Scripting, Memory Leaks, Weak Cryptography, etc.). Additional open-source static code analyzers are also utilized (e.g., CppCheck and FindBugs) through dedicated SonarQube plugins.

Subsequently, the low-level static analysis alerts are fed to the *Security Measures Computation* mechanism, which aggregates them to compute the high-level security measures. IoT-specific security models are utilized (leveraging concepts from state-of-the-art security and quality models [8], [9]) to determine (i) the high-level security measures that should be computed, and (ii) which low-level static analysis results should be aggregated (and in what way) to quantify those

[3]https://www.sonarqube.org/

measures. The output of SEF is a report containing the detailed results of the analysis, and particularly: (i) the calculated high-level security measures, and (ii) the low-level static analysis alerts that were utilized for computing those measures.

Security Alerts Criticality Assessor: Although effective in detecting security issues, static analysis is known to produce long lists of alerts, most of them not being critical from a security viewpoint. This hinders its practicality, since developers often have to go through the tedious process of triaging the alerts to detect those that correspond to critical security issues.

In an attempt to address the aforementioned problem, we propose a novel mechanism for assessing the criticality of security-related static analysis alerts [13]. In particular, we developed a self-adaptive technique, the *Security Alerts Criticality Assessor (SACA)*, for classifying and prioritizing security-related static analysis alerts based on their criticality, by considering information retrieved from (i) the alerts themselves, (ii) vulnerability prediction (see Section IV-B2), and (iii) user feedback. The proposed technique is based on machine learning models, particularly on neural networks, which were built using data retrieved from static analysis reports of real-world software applications. The high-level overview of the tool is presented in Figure 5.

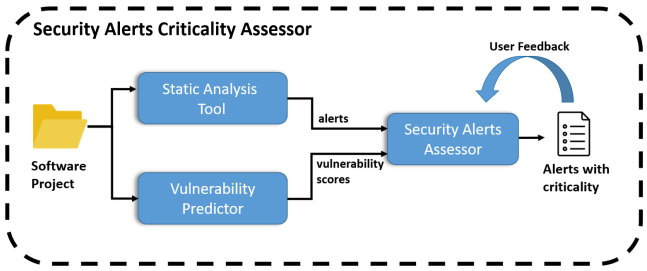

Fig. 5: Overview of the Security Alerts Criticality Assessor

As can be seen by Figure 5, a software project is provided as input to the system and subsequently, security-specific static analysis is employed to retrieve the alerts that are relevant to this project. In addition, vulnerability prediction models are employed to compute the *vulnerability scores* (see Section IV-B2) of its software components, which reflect the likelihood of each component to contain vulnerabilities. Then, the alerts along with the *vulnerability scores* are provided as input to the *Security Alerts Assessor (SAA)*, i.e., a neural network that assesses how critical each one of the alerts is from a security viewpoint. More specifically, for each one of the received alerts, it reports (i) a *criticality flag* (i.e., a binary value between 0 and 1, with 1 denoting that the corresponding alert is considered critical), and (ii) a *criticality score* (i.e., a continuous value in the [0,1] interval denoting how likely it is for the alert to be security-critical). Hence, developers can start their refactoring activities by fixing alerts that have higher criticality, increasing in that way the probability of detecting and mitigating actual vulnerabilities.

As illustrated in Figure 5, the user can also correct the outputs of the model, and the SAA can be retrained on demand, based on user feedback. In that way, the SAA adapts to the specific characteristics of the software product to which it is applied to provide more accurate assessments.

2) Vulnerability Prediction: Vulnerability Prediction is responsible for the identification of security hotspots, i.e., software components (e.g., classes) that are likely to contain critical vulnerabilities. For the identification of potentially vulnerable software components, vulnerability prediction models (VPM) are constructed, which are mainly machine learning models that are built based on software attributes retrieved primarily from the source code of the analyzed software (e.g., software metrics, text features, etc.). The results of the vulnerability prediction models are highly useful for developers and project managers, as they allow them to better prioritize their testing and fortification efforts by allocating limited test resources to high-risk (i.e., potentially vulnerable) areas.

a) Component-level Vulnerability Prediction: Existing vulnerability prediction models focus on the intrinsic characteristics of the analyzed software component, and particularly on attributes of its source code, in order to judge whether it is vulnerable or not. Among the different attributes that have been examined in the literature, those that are derived through text mining have demonstrated the most promising results [10], [14]. To this end, as part of the *Software Security Assurance* module of the SSD Platform, we developed vulnerability prediction models based on deep learning, utilizing as input the sequences of word tokens that reside in the source code of the component, and word embedding vectors for their effective representation [15]. A high-level overview of the proposed models is illustrated in Figure 6.

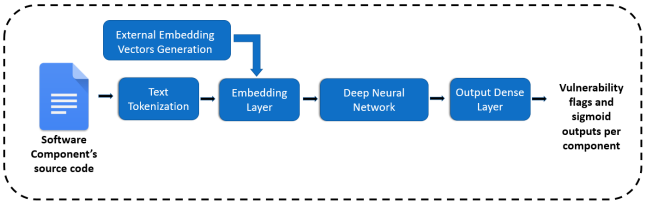

Fig. 6: Overview of Component-level Vulnerability Prediction

As can be seen by Figure 6, the source code of a software component is provided as input to the model and subsequently, text tokenization is applied in order to extract the word tokens and construct their corresponding sequence. Then, since Deep Neural Networks (DNN) operate on numerical values, the token sequences need to be properly transformed into numerical vectors. For this purpose, word embedding vectors are utilized. Word embedding refers to the representation of words, in the form of a real-valued vector that encodes the meaning of the word in such a way that words that are close in the vector space are expected either to have similar meanings or to be in close proximity in the source code. Our mechanism utilizes two popular word embedding algorithms to generate the word embedding vectors, namely word2vec[4] and fast-text[5].

[4]https://radimrehurek.com/gensim/models/word2vec.html
[5]https://radimrehurek.com/gensim/models/fasttext.html

The derived word embedding vectors constitute the word embedding layer, i.e., the input layer of the DNN, which is the core element of the model. The output of the model is a *vulnerability flag* (i.e., a binary value between 0 and 1) indicating whether the given component is potentially vulnerable (i.e., 1) or not (i.e., 0), and a *vulnerability score* (i.e., a continuous value within the [0,1] interval) denoting how likely it is for the component to be vulnerable. The developers can use this information in order to decide where to focus their testing and refactoring activities, starting, for instance, from those components that have higher *vulnerability score*.

b) System-wide Vulnerability Prediction: Section IV-B2a summarized existing models to predict software vulnerability based solely on analyzing the software component's source code to indicate whether it may contain a critical vulnerability. Such models do not account for the overall architecture of the application, which is an important issue when software consists of multiple interconnected components; i.e. the vulnerability of a component can affect other components, regardless of their individual vulnerability. To the best of our knowledge, vulnerability prediction models including such interdependencies have not been published.

Thus we present a novel mechanism for system-wide vulnerability prediction based on Adversarial Random Neural Networks (ARNN) [16] with learning [17] previously used for network cyber-attack detection [18]. The ARNN takes into account (i) the vulnerability score of each software component derived from component-level VPMs, and (ii) the interconnections among components of the software under analysis, to compute the resulting system-level vulnerability likelihood of each component, as summarized in Figure 7.

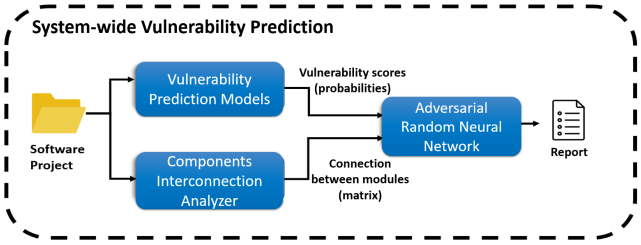

Fig. 7: Overview of the System-wide Vulnerability Prediction

Figure 7 shows how the source code of the application under analysis is inputed to the proposed mechanism. First, the text mining-based vulnerability prediction models of Section IV-B2a are used to derive their *vulnerability scores* denoting how likely it is for each software component to contain vulnerabilities in its source code. Then a *Component Interconnection Analyzer* is used to detect interconnections between components based on function calls, providing an Adjacency Matrix. The Adjacency Matrix and the individual vulnerability scores of components are inputed to the ARNN [16], which outputs the Likelihood Ratio indicating how likely it is for each interconnected software component to be vulnerable, based both on its own vulnerability and the effect of other components in the application. Thus the proposed mechanism combines vulnerability scores of individual components, with a system-level data regarding the interconnected components.

V. Conclusion

The present paper provides an overview of the Software Security-by-Design (SSD) Platform, developed within the context of the IoTAC project. It describse the main novel mechanisms proposed and developed in IoTAC, including novel security monitoring and optimization mechanisms developed in several phases of the overall SDLC, regarding the Design, Requirements, Coding, and Testing phases which are at the core of the SSD Platform. They constitute the basis on which the Software Security Certification module operates to provide a broad evaluation of the security level of an IoT software application, which may be issued as a "pseudo-certificate". In the rest of the IoTAC project, emphasis will be given on building the Software Security Certification module, and further improving the mechanisms described in this paper.

References

[1] J. Soldatos, *Security Risk Management for the Internet of Things: Technologies and Techniques for IoT Security, Privacy and Data Protection*. Now Publishers, 2020.

[2] ISO/IEC, *ISO/IEC 30141:2018 - Internet of Things (IoT) — Reference Architecture*. ISO/IEC, 2018.

[3] M. Ramachandran, "Software security requirements engineering: State of the art," in *Int. Conf. on Global Security, Safety, and Sustainability*.

[4] T. Diamantopoulos and A. L. Symeonidis, *Mining Software Engineering Data for Software Reuse*. Springer Nature, 2020.

[5] T. Diamantopoulos and A. Symeonidis, "Enhancing requirements reusability through semantic modeling and data mining techniques," *Enterprise information systems*, 2018, publisher: Taylor & Francis.

[6] E. Yu, "Modeling Strategic Relationships for Process Reengineering." *Social Modeling for Requirements Engineering*, no. 2011.

[7] G. McGraw, "Software security," *IEEE Security & Privacy*, 2004.

[8] U. Dayanandan and V. Kalimuthu, "Software architectural quality assessment model for security analysis using fuzzy analytical hierarchy process (fahp) method," *3D Research*, 2018.

[9] I. Heitlager, T. Kuipers, and J. Visser, "A practical model for measuring maintainability," in *6th international conference on the quality of information and communications technology*. IEEE, 2007.

[10] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014.

[11] D. Tsoukalas, M. Siavvas, M. Mathioudaki, and D. Kehagias, "An ontology-based approach for automatic specification, verification, and validation of software security requirements: Preliminary results," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, 2021.

[12] M. Howard, *Writing secure code*. Microsoft Press, 2003.

[13] M. Siavvas, I. Kalouptsoglou, D. Tsoukalas, and D. Kehagias, "A self-adaptive approach for assessing the criticality of security-related static analysis alerts," in *International Conference on Computational Science and Its Applications*. Springer, 2021, pp. 289–305.

[14] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet," *IEEE Transactions on Software Engineering*, 2021.

[15] I. Kalouptsoglou, M. Siavvas, D. Kehagias, A. Chatzigeorgiou, and A. Ampatzoglou, "An empirical evaluation of the usefulness of word embedding techniques in deep learning-based vulnerability prediction," in *EuroCybersec2021*, 2021.

[16] E. Gelenbe and M. Nakip, "The Adversarial Random Neural Network and Botnet attack detection," *Submitted for publication*, 2021.

[17] E. Gelenbe, "Learning in the recurrent random neural network," *Neural Computation*, no. 1, pp. 154–164, 1993.

[18] M. Nakip and E. Gelenbe, "Mirai botnet attack detection with auto-associative dense random neural networks," in *2021 IEEE Global Communications Conference*. IEEE Communications Society, 2021.