# Evaluating and Improving the Internal Security of OPC-UA based Software Applications

Marija Jankovic[a], Miltiadis Siavvas[a], Dimitrios Tsoukalas[a] and Dionysios Kehagias[a]

[a] *Centre for Research and Technology Hellas, 6th km Harilaou - Thermi, Thessaloniki, 57001, Greece*

**Abstract**

High complexity, extensibility, and interconnectivity of industry 4.0 software systems introduce critical software security issues. Open Platform Communications Unified Architecture (OPC-UA) standard specification highlights the need to provide adequate support for the implementation of Confidentiality, Integrity, and Availability (CIA triad) aspects. However, developers and engineers often overlook these critical security concerns, leading to software systems bundled with severe vulnerabilities. The exploitation of a single vulnerability may lead to far-reaching consequences to the compromised software's owing enterprise. Measuring and evaluating software security is crucial for secure software development. The paper gives the theoretical and technical background of the Quantitative Security Assessment and Vulnerability Prediction services, which are part of the SDK4ED Dependability toolbox. Moreover, it presents the results of the performed security evaluation of an OPC-UA based open-source application. Finally, it discusses the refactoring recommendations on the source-code level, leading to improved security.

**Keywords 1**

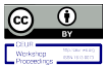
Security Assessment, Vulnerability Prediction, Static Analysis, OPC-UA

## 1. Introduction

OPC-UA serves as the de-facto standard for data interoperability and collaboration between various IoT, M2M, and Industry 4.0 devices in local and distributed settings [1]. The topic of security, which gained a lot of interest in the automation domain, is recognized as one of the fundamental requirements for successfully implementing the OPC-UA standard [2, 4]. Today's production systems are becoming decentralized and relying on a distributed supply chain in a global environment. A security incident in such an organizational environment could have much more significant consequences than systems that act in isolation (i.e., local network within the plant). [6].

One of the controversially discussed topics in the literature is determining the appropriate level of security [6, 3]. This is a challenging research topic and requires investigation of various requirements, such as a targeted organizational environment, a technology used, etc. However, to investigate how much security a system needs, one should evaluate its current security level. Authors in [7] point out that "you cannot control something you cannot measure". Research in the field of internal software quality assessment is still in the early development stage, and there is no well-accepted technique for overall internal software security evaluation [8].

Traditionally, software security is considered an afterthought in the software development lifecycle (SDLC), as it is usually added and assessed after the implementation of software products during the Deployment and Operation phase. However, there is an observed shift towards the Security by Design paradigm, which suggests integrating security from the early SDLC development phases, such as requirement elicitation and coding [10]. To produce highly secure applications from the beginning, it is important to measure and improve their internal security level continuously. In the context of the ongoing SDK4ED Horizon 2020 project[2], we have developed a novel Security Assessment Model

Proceedings of the Workshops of I-ESA 2020, 17-11-2020, Tarbes, France
EMAIL: jankovicm@iti.gr (A. 1); siavvasm@iti.gr (A. 2); tsoukj@iti.gr (A. 3); diok@iti.gr (A.4)
ORCID: 0000-0002-3600-3014 (A. 1); 0000-0002-3251-8723 (A. 2); 0000-0001-9986-0796 (A. 3); 0000-0002-6912-3493 (A.4)



[2] https://sdk4ed.eu/

(SAM) that facilitates quantitative security evaluation of software applications based on various static analysis alerts and software metrics. Besides, we propose a novel Vulnerability Prediction Model for the prediction of potential vulnerabilities.

The following case study presents an approach to validate the capability of the SDK4ED Dependability toolkit to perform Quantitative Security Assessment and Vulnerability Prediction of OPC-UA-based open-source applications. In this context, Section 2 gives a brief background of quantitative security assessment and vulnerability prediction. Section 3 explains the details of the evaluation procedure. Section 4 elaborates on the results achieved. Finally, in Section 5, we give conclusions and discuss future work.

## 2. Background

A software security assessment is considered in the literature as a sub-field of software quality evaluation [9]. The Quantitative Security Assessment (SAM) model combines heterogeneous security metrics with the goal of providing a quantifiable expression of software security [5]. The hierarchical decomposition of the model elements supports fine-grained security assessment at the various levels of abstraction as illustrated in Figure 1 [8]. The model is designed based on the ISO/IEC 25010 general recommendation to hierarchically decompose the security quality into a set of main security characteristics and corresponding sub-properties, which are linked with specific measures [16]. In particular, the SAM systematically aggregates a set of low-level security indicators to produce a high-level score that reflects the internal security level of analyzed application. There are four different layers namely *security*, *characteristics*, *properties* and *measures*.

***Characteristics Layer*** consists of three security characteristics, namely *Confidentiality*, *Integrity*, and *Availability*, which together form the CIA triad of information security [11]. The CIA triad characteristics present the core objectives of the OPC-UA security model [1, 4]. ***Properties Layer*** comprises 4 properties (i.e. *Complexity*, *Cohesion*, *Coupling*, and *Encapsulation*) quantified through software metrics, which are calculated using CKJM Extended software metrics tool [12]. Besides, this layer includes 7 vulnerability categories (i.e. *Null Pointer*, *Assignment*, *Exception Handling*, *Resource Handling*, *Logging*, *Misused Functionality*, and *Synchronization*). These vulnerability categories are calculated using static code analyzers such as PMD, which is included in both OWASP and NIST lists of recommended static analysis tools [13, 14, 15]. Each property of the ***Metric Layer*** is quantified by a single code-level measure. The values of these measures are used along with a set of thresholds in order to assign ratings (i.e. scores) to a group of higher-level properties (e.g. Complexity). These ratings are then aggregated using a weighted average scheme in order to calculate the ratings of a set of security characteristics (e.g. Confidentiality). Finally, the produced ratings are aggregated again in order to calculate the overall ***Security level*** (i.e., the *Security Index*) of the software product under analysis. This score resides in the [0,1] interval, where 0 denotes bad security, whereas 1 denotes sufficient security [5]. The SAM supports the evaluation of the security level of software applications written in Java, C, and C++ programming languages.

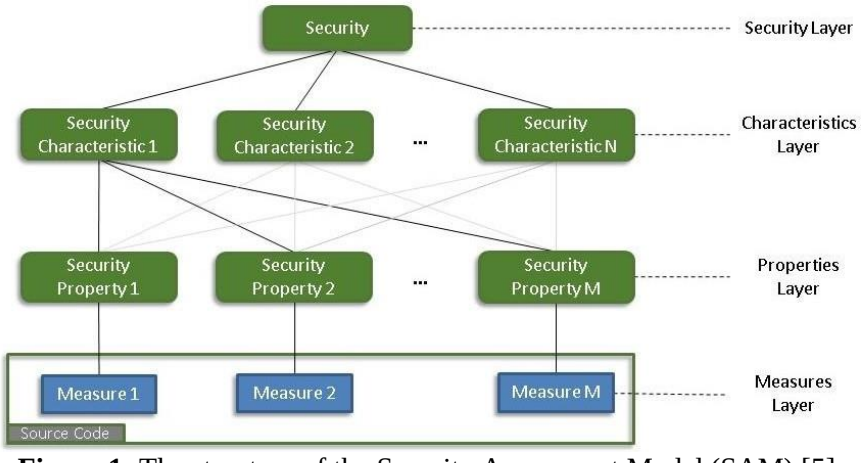



**Figure 1:** The structure of the Security Assessment Model (SAM) [5]

Vulnerability prediction is a relatively new research area in the field of software security, which focuses on identifying potential indicators of software security risks and building corresponding vulnerability predictors. Most of the research efforts related to vulnerability prediction can be classified into one of the three groups: (i) static analysis-alerts, (ii) software metrics, and text-mining features [11]. Within the SDK4ED project, two Vulnerability Prediction Models (VPM) based on deep learning and text mining have been developed to support the distinctive needs of software applications written in Java and C/C++ programming languages.

## 3. Evaluation procedure

The overall evaluation procedure comprises two main steps: (1) Quantitative security assessment, and (2) Vulnerability Prediction of software applications retrieved from the open-source OPC-UA Foundation Git repository [18]. SAM and VPM models are implemented as REST API services of the Dependability module of the SDK4ED platform [17]. The services can be invoked both through the SDK4ED main dashboard and individually through a dedicated API by providing the following parameters: (i) project (the URL to an online repository); (ii) language (java or C/C++); and (iii) inspection parameter (yes/no).

The output of the QSA web service is a JSON file containing the security assessment report, which includes the security index, the security score of the model properties and characteristics, and the detailed static analysis results (if value of the inspection parameter is set to 'yes'). The inspection parameter is useful as it allows the selection of the level of detail of the produced security assessment report based on the user needs. By setting the value of this parameter to "no", raw results, which are long lists of static analysis alerts will be omitted leading to the production of lightweight and comprehensive report. The output of the Vulnerability Prediction web service is a JSON file with the vulnerability report, which contains (i) the names of the analyzed source code files of the application, (i) their vulnerability status as produced by the applied model (i.e., 1 if they are potentially vulnerable and 0 if they are potentially clean), and (iii) the probability of containing vulnerabilities. The results are discussed in more detail in the following section.

## 4. Results

The purpose of this section is to demonstrate the results of the analyzed open-source OPC-UA application and indicate the benefits of security monitoring during the early stages of an SDLC, such as development and testing. As shown in Figure 3, the overall security index of the analyzed UA-Java-Legacy application is shown both in numerical (70%) and in discrete format (4 stars) [18]. Besides, two radar charts are provided, showing the individual scores of the model properties and characteristics. From the radar chart in Figure 3, it is evident that all properties received a high score apart from *Null_Pointer* that received a deficient score (0.04), and *Synchronization* which received an average score (0.56).


**Figure 2:** The Security Index


**Figure 3:** The scores of a security properties and characteristics of the UA-Java-Legacy application

Developers and project managers can leverage this information for deciding where to focus their testing and refactoring efforts. For example, they should start their refactoring activities by fixing issues relevant to the *Null_Pointer* property. Besides, the user has an opportunity to filter specific property and receive a detailed list of all relevant issues ranked based on their severity. The resulting table contains useful information regarding the source code file to which issues belong, and the exact line of code, as illustrated in Figure 4. In that way, developers can focus on fixing the topmost priority issues.

Show entries 10 Search Search

| Rule Name | RuleSet Name | Package Name | Class Name | Line of Code | Priority |
|---|---|---|---|---|---|
| NullAssignment | Controversial | org.opcfoundation.ua.application | src\application\Application.java | 377 | 3 |
| NullAssignment | Controversial | org.opcfoundation.ua.application | src\application\Application.java | 381 | 3 |

**Figure 4:** Overview of static analysis alerts on the SDK4ED Dashboard

As far as the Vulnerability Prediction service results are concerned, these are presented on the SDK4ED dependability dashboard both in graphical and tabular forms. Figure 5 presents part of the vulnerability prediction results of the analyzed UA-AnsiC-Legacy application. The results are illustrated in the form of a heatmap, where each rectangle corresponds to a specific class of the analyzed source code project. The color of the rectangle denotes the probability of the corresponding source code file to contain vulnerabilities. For example, the darker shades indicate the higher probability that associated source code will contain vulnerabilities. This visualization is useful for the developers and software managers, as it allows them to pinpoint the hotspots of the software project easily. Besides, the Vulnerability Prediction service supports the generation of more detailed reports indicating actual probability score.

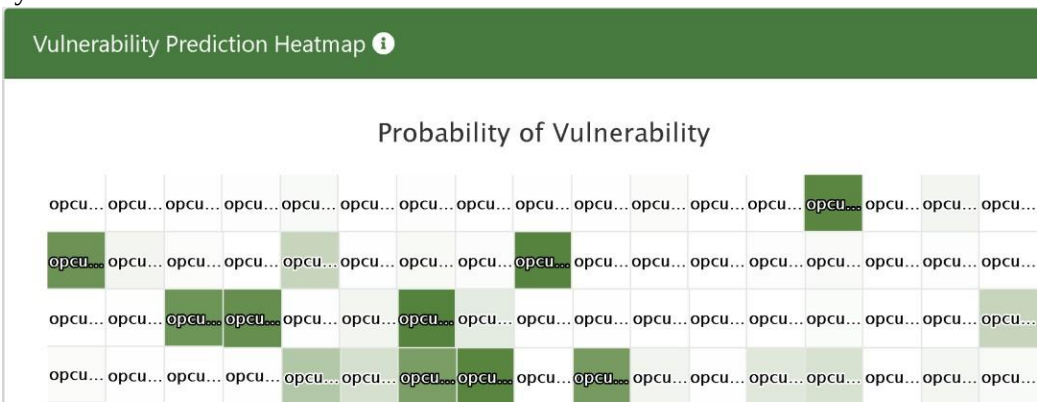



**Figure 5:** A heatmap visualizing the vulnerability prediction results of the UA-AncyC-Legacy application

## 5. Concluding discussion

In this paper, we demonstrated that proposed SAM and VP models, can be applied for evaluation of the OPC-UA open source applications written in Java programming language. The conducted case study verified the ability of proposed models to analyze the selected software applications, while they also led to practical recommendations on how to improve overall internal security. The overview of security issues at different level of granularity was helpful to define correct refactoring policy. Overall Security Index, and score of key characteristics is intuitive for higher management, while developers can benefit from detailed analytical overview of all issues and prioritization and localization in source code. The visualization heatmap is expected to facilitate the prioritization of testing and fortification efforts of the software developers, by allocating usually limited test resources to high risk areas which are potentially vulnerable. For example, more exhaustive security testing should be allied to source code files that are more likely to contain vulnerabilities. In the future we are planning to evaluate the proposed models in a real Industry 4.0 production environment.

ACKNOWLEDGMENT

Work reported in this paper has received funding from the European Union Horizon 2020 research and innovation programme under grant agreement No. 780572 (project: SDK4ED).

## 6. References

[1] OPC Foundation, Practical Security Recommendations for building OPC UA Applications, Version 3, 2018. URL: https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Security-Advise-EN.pdf

[2] Federal Office for Information Security (BSI), OPC UA Security Analysis, 2017. URL: https://opcfoundation.org/wp-content/uploads/2017/04/OPC_UA_security_analysis-OPC-F-Responses-2017_04_21.pdf

[3] P. Cheremushkin, S. Temnikov, Kasperski LAB ICS CERT, OPC UA Security Analysis, 2018. URL: https://ics-cert.kaspersky.com/media/KL_OPCUA_MAY_2018_EN.pdf

[4] OPC Foundation, OPC UA Specification: Part 2 – Security Model, 2008. URL: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/

[5] M. Siavvas, Static Analysis for Facilitating Secure and Reliable Software, PhD thesis, ICL London, 2019

[6] W. Mahnke, S.H. Leitner, M. Damm, OPC Unified Architecture, Springer, Berlin, Heidelberg, 2009. DOI : https://doi.org/10.1007/978-3-540-68899-0

[7] P. K. Manadhata and J. M. Wing, Measuring a Systems Attack Surface, 2004. URL : https://www.cs.cmu.edu/~wing/publications/tr04-102.pdf

[8] M. Siavvas, K. C. Chatzidimitriou, and A. L. Symeonidis, QATCH - An adaptive framework for software product quality assessment, Expert Systems with Applications, vol. 86, (2017). DOI : 10.1016/j.eswa.2017.05.060

[9] N. Munaiah, F. Camilo, W. Wigham, A. Meneely, and M. Nagappan, Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project, Empir. Softw. Eng., 1305–1347 (2017). https://doi.org/10.1007/s10664-016-9447-3

[10] S. Zafar, M. Mehboob, A. Naveed, and B. Malik, "Security quality model: an extension of Dromey's model," Softw. Quality Journal, vol. 23, no. 1, (2015) DOI : 10.1007/s11219-013-9223-1.

[11] J. Andress, The basics of information security: understanding the fundamentals of InfoSec in theory and practice. Waltham, MA: Syngress, (2014). ISBN-13: 978-0128007440

[12] CKJM extended - An extended version of Tool for Calculating Chidamber and Kemerer Java Metrics. 2011. URL : http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/

[13] PMD An exstensible cross- language static code analyzer. 2020. URL : https://pmd.github.io/

[14] OWASP, Source Code Analysis Tools, 2020. URL : https://owasp.org/www-community/Source_Code_Analysis_Tools

[15] NIST, Source Code Security Analyzers, 2020. URL : https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html
[16] ISO/IEC 25010 :2011 Systems and Software Quality Requirements and Evaluatin (SQuaRE), (2017). https://www.iso.org/standard/35733.html
[17] M. Siavvas, D. Tsoukalas, C. Marantos, A.A. Tsintzira, M. Jankovic, D. Soudris, A. Chatzigeorgiou, and D. Kehagias, The SDK4ED Platform for Embedded Software Quality Improvement-Preliminary Overview, in Proceedings of International Conference on Computational Science and Its Applications (pp. 1035-1050). Springer, Cham. 2020. DOI : https://doi.org/10.1007/978-3-030-58811-3_73
[18] OPC Foundation, Build OPCUA applications with Java, 2018. URL: http://opcfoundation.github.io/UA-Java-Legacy/