


Automated Java Challenges' Security Assessment for Training in Industry – Preliminary Results

Luís Afonso Casqueiro ✉ 

University Institute of Lisbon, (ISCTE-IUL), ISTAR, Portugal

Tiago Espinha Gasiba ✉ 

Siemens AG, Munich, Germany

Maria Pinto-Albuquerque ✉ 

University Institute of Lisbon (ISCTE-IUL), ISTAR, Portugal

Ulrike Lechner ✉ 

Universität der Bundeswehr München, Munich, Germany

Abstract

Secure software development is a crucial topic that companies need to address to develop high-quality software. However, it has been shown that software developers lack secure coding awareness. In this work, we use a serious game approach that presents players with Java challenges to raise Java programmers' secure coding awareness. Towards this, we adapted an existing platform, embedded in a serious game, to assess Java secure coding exercises and performed an empirical study. Our preliminary results provide a positive indication of our solution's viability as a means of secure software development training. Our contribution can be used by practitioners and researchers alike through an overview on the implementation of automatic security assessment of Java CyberSecurity Challenges and their evaluation in an industrial context.

2012 ACM Subject Classification Security and privacy → Software security engineering; Security and privacy → Web application security; Applied computing → Computer-assisted instruction; Applied computing → E-learning; Applied computing → Interactive learning environments

Keywords and phrases Education, Teaching, Training, Awareness, Secure Coding, Industry, Programming, Cybersecurity, Capture-the-Flag, Intelligent Coach

Digital Object Identifier 10.4230/OASICS.ICPEEC.2021.10

Funding *Maria Pinto-Albuquerque:* This work is partially financed by national funds through FCT - Fundação para a Ciência e Tecnologia, I.P., under the projects FCT UIDB/04466/2020 and UIDP/04466/2020. Furthermore, the first and third author thank the Instituto Universitário de Lisboa and ISTAR, for their support.

Acknowledgements The authors would like to thank all the survey participants for taking part in this preliminary study, and for their helpful and constructive feedback. Furthermore, the authors would like to thank the hosting organization for enabling the study to take place.

1 Introduction

Over the last years, the number of cybersecurity incidents has been continually rising. According to the US department of homeland security [7], the root cause of about 90% of these incidents is due to poor software quality, which results in software vulnerabilities. A recent large-scale study by Patel et al. [17] has shown that more than 50% of software developers cannot identify vulnerabilities in source code. These facts pose serious issues in the industry, especially for companies that deliver products to critical infrastructures. Companies can use several methods to improve software quality, such as code reviews and static application security testing (SAST). Another possibility is to address the human factor, i.e., the software developers, through education on secure coding.



© Luís Afonso Casqueiro, Tiago Espinha Gasiba, Maria Pinto-Albuquerque, and Ulrike Lechner; licensed under Creative Commons License CC-BY 4.0

Second International Computer Programming Education Conference (ICPEEC 2021).

Editors: Pedro Rangel Henriques, Filipe Portela, Ricardo Queirós, and Alberto Simões; Article No. 10;

pp. 10:1–10:11



OpenAccess Series in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

10:2 Automated Security Assessment of Java CyberSecurity Challenges

In this work, we explore raising awareness on secure coding using serious games. Previous work by Gasiba et al. [10] identifies serious games as a viable alternative to raise secure coding awareness of software developers in the industry. However, their work does not address Java programmers. Java is not only widely used in the industry and taught at universities, but it is also in the top-3 of the programming languages that have the most open source vulnerabilities [18].

In this work, we propose and implement a method to perform an automatic evaluation of the security level of Java challenges. This process generates hints for the players when the code contains vulnerabilities. Our platform is embedded in a serious game called CyberSecurity Challenges [11] that has the goal of raising awareness of software developers in the industry. This game is inspired in the capture-the-flag genre of games and includes several types of challenges, both offensive and defensive in nature. Once a team solves one of the challenges they receive a flag earning them a certain amount of points. In the end the team with most points wins the Cyber Security Challenges event. No one wins if no one passes all the tests. However, the main goal of the game is to raise awareness of secure coding, and winning the game by collecting points serves only as an incentive for the individual players and teams to actively participate in the game.

The designed java challenge's artifact, in the present work, allows for a fully automatic interaction between players through the hint system. The hints are designed to raise awareness of the Carnegie Mellon's Java secure coding guidelines [22]. Our research questions are:

RQ1 How to automatically assess the level of security of a player's Java code?

RQ2 Which open-source tools can be used to assist the Java security code assessment?

RQ3 How do players perceive the platform as a means to learn Java secure coding?

The main contribution of this work is insight for practitioners and researchers who wish to implement Java secure coding challenges with automatic hint generation. Moreover, this work also provides a relevant method to automatically assess the security levels of Java code, a list of relevant tools and an analysis in the industry setting. Additionally, the empirical study and preliminary results provide a further contribution to knowledge, and its raw results are provided in Zenodo [14] to enable further research.

The present work is organized as follows; section 2 discusses previous relevant work, section 3 describes how to perform automatic security assessment of Java challenges, and the tools we use. In section 4, results of a preliminary study in the industry is presented, and section 5 concludes this work.

2 Related Work

In [15], Meng et al. identified several security-related problems that Java software developers experience when developing software. In particular, they conclude that developers do not understand the security implications of their coding decisions. Also, developers tend to search online forums for answers. However, Fischer et al. [9] have shown that such information can be outdated and possibly even wrong. Oliveira et al. [16] discuss the lack of awareness by software developers of the specification and proper usage of application programming interfaces (API). They conclude that wrong usage of APIs can lead to security vulnerabilities. Gasiba et al. identify that software developers in the industry lack awareness [13] of secure coding guidelines, e.g., [22]. The result of their study is in agreement with a recent large-scale survey, with more than 4000 software developers, by Patel et al. [17] that concludes that more than 50% of software developers cannot identify vulnerabilities in source code.

One possible way to raise awareness of secure coding is through serious games. Dörner et al. [8] define serious games as a game that is *designed with a primary goal and purpose other than pure entertainment*. Culliane et al. [6] argue that these types of games can be educational and fun, and Sorace et al. [24] conclude that these can be an attractive approach to improve awareness. Graziotin et al. [12] show that *happy developers are better coders*. Furthermore, Gasiba et al. developed C/C++ CyberSecurity Challenges for industrial software developers [11]. In this work, we adapted this platform to provide Java CyberSecurity Challenges.

Several serious games have been developed with success. Bakan et al. [1] provide an overview of current research in game-based learning and teaching environments. They conclude that serious games can be an effective teaching tool in terms of students' achievements and retention. In a similar work, Cardoso et al. [2] propose to integrate a Virtual Programming Lab in Moodle to provide a platform for students to edit, compile, run, debug and evaluate programs. However, their work is performed in academia, focuses on teaching Java programming to undergraduates, and does not focus on secure coding.

To design a serious game for secure programming in Java, which produces automatic hints, a method to automatically evaluate a player's code in terms of cybersecurity vulnerabilities needs to be developed. However, to the best of our knowledge, previous literature does not address these combined issues.

The present work uses and is embedded in a study that uses Action-Design Science methodology by Sein et al. [21]. This research methodology addresses real-world problems in organizations through meaningful actions which are informed by cooperation with academia.

3 Automated Security Assessment of Java Challenges

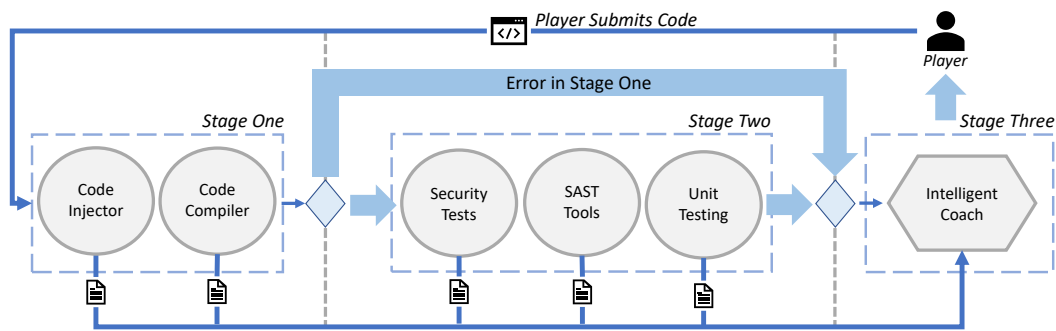
This section briefly describes the automated security assessment process, how the implementation-related security problems are addressed, and describes an empirical study performed in the industry.

3.1 Automated Security Assessment Process

A Java challenge consists of the following phases. First, the player is presented with a challenge (programming exercise) through a web interface containing at least one cybersecurity vulnerability. The player's task is to identify the vulnerability in the code and rewrite it such that the rewritten code still implements the challenge's required functionality. When the player is satisfied with the rewritten code, he or she submits it to the backend. At this point, the backend performs several analysis steps to the submitted code to automatically assess it in terms of vulnerabilities and functionality. If the code contains no vulnerabilities and is functionally correct, the player has won the challenge. Otherwise, the backend generates a hint to send back to the player to help them solve the challenge. The hint's generation is based on the player's code and the various analysis steps performed during code analysis. Furthermore, the hints are implemented using a laddering technique [19, 11].

Figure 1 details the various stages undertaken during the submitted code's automated assessment. These stages are composed of several steps. The first stage includes two steps: code injection and compilation. When needed, the players' code is modified through code injection. This step might be required to insert additional functionalities, such as new classes into the project, through import statements without the player's knowledge. Next, the resulting code is compiled. If any errors are encountered, the process jumps to stage three. In the absence of compiler errors, the process advances to stage two.

10:4 Automated Security Assessment of Java CyberSecurity Challenges



■ **Figure 1** Automatic Assessment Process.

There are two major possible approaches to stage two. The tools employed can either run in parallel or sequentially. In this case we used a sequential process due to the fact that the amount of tools used plus the correspondent execution time does not justify a parallel approach. This means that the difference in the response time does not substantially improve from one approach to the other.

This stage consists of code tests using three different testing methods: security tests, static code analysis, and unit tests.

Unit testing is performed to ensure that the player’s code works according to the challenge’s specifications. Security tests are dynamic unit tests that mimic malicious user actions and input to exploit the exercise’s vulnerability. Static-application security testing (SAST) tools provide a report based on analysis of the player’s code that includes code vulnerability issues. These tools however can produce wrong evaluations of the given code.

■ **Table 1** SAST Tools Outcomes.

Type	Description
True Positive	Code has a vulnerability and the tool flags it
True Negative	Code doesn’t have a vulnerability, and the tool does not flag it
False Positive	Code doesn’t have a vulnerability, but the tool flags it
False Negative	Code has a vulnerability, and the tool doesn’t flag it

Table 1 shows the possible outcomes of using SAST tools. Of the four possibilities only two represent the desired functionality of these tools, True Positive and True Negative. In our work, we only consider a sub-set of the results reported by the tools, in order to minimize the number of false positives and false negatives in our security assessment.

Table 2 summarizes the tools that we have used in this stage and their mapping to the testing methods. These tools were selected based on the capability to answer the stage two needs of testing. For the SAST tools we gathered industry recognized free and open-source software and then filtered by the amount of security related errors. The JUnit tool, a framework widely used in industry, was implemented for unit-testing of the submitted code. This tool ensures that the player’s code works according to the specifications and goals of the challenge. The unit-tests will flag issues like empty class (where no security vulnerabilities are found), giving the proper feedback to the player. JUnit is also used in security-testing of the code. This tool provides extreme values in the functions call in order to try and trigger any potential issue present in the program. JavaFuzzer and Spoon are other tools utilized

■ **Table 2** Analysis Tools.

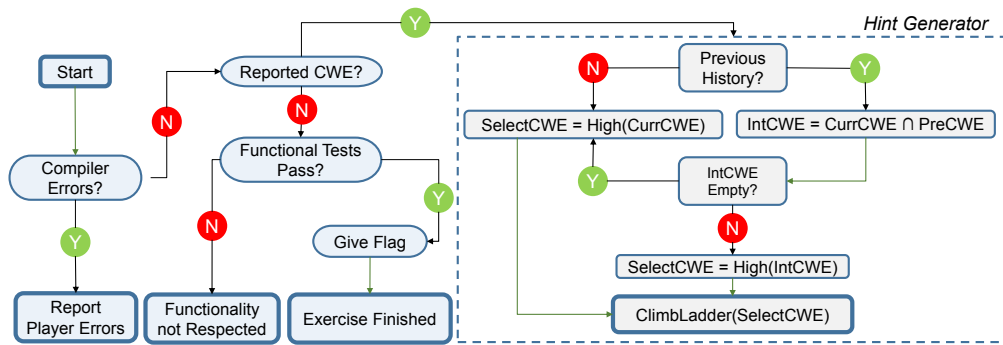
Tools	Unit Tests	Security Tests	SAST Tools
JUnit	•	•	
JavaFuzzer		•	
Spoon		•	
Self-Developed Tools		•	
SonarQube			•
SemGrep			•
FB-Infer			•
SpotBugs			•
PMD			•
JBMC			•

for security-testing. These tools provide, among others, a series of tests that stimulate the given code by creating random inputs in order to try and trigger potential errors in the given program [20].

The employment of several different testing tools and methods offers a greater variety of results. Although many times different testing tools flag the same errors, there are also several cases where only specific tools can spot certain security vulnerabilities. This means that the tools used can sometimes overlap but also complement each others in the results of the analysis, leading to a more complete scrutiny of the player's code. These tools are used to generate security-related findings of the submitted code. The tools' findings are mapped to a PASSFAIL boolean value and to a specific common weakness enumeration (CWE) [3]. The mapping to individual CWEs as well as its ranking was done according to our experience in cybersecurity. Note that, since the user is free to change the exercise's code at will, potentially malicious code will be executed in stage two. Therefore, to prevent malicious players from abusing and breaching our system (intentionally or non-intentionally), the security tests and unit tests run in a time-limited sandbox. The time-limitation protects against infinite loops, and the sandbox protects against remote-code execution. For more information, we refer the reader to [11].

The last stage consists of an artificial intelligence engine that creates relevant hints for the player if the challenge is not solved. If the challenge is solved, the last stage awards points to the player. Hints are generated based on a CWE selected from the previous steps' results and prior history, using a laddering technique. The selection of the CWE, which is supplied to the laddering technique, is shown in figure 2.

In case of compiler errors, no CWE is selected, and a report is sent back to the player, which includes the compiler error message. If the code compiles with no errors but no CWE is reported from the tools, the algorithm checks the functional tests' results. If all the tests pass, the player wins the game, is given a flag in recognition, and the exercise is finished. If at least one test fails, the player receives feedback based on the failed functional test case. This feedback states that the desired code functionality is not respected. However, when the code has security issues (i.e., at least one CWE is reported), the hint generator starts. The hint-generation checks if there have been previous hints given to the player based on the prior history. If the prior history is empty, the hint will be generated based on the highest-ranking CWE found out of all the reported CWEs. This CWE ranking was pre-determined by us based on our experience.



■ **Figure 2** Intelligent Coach’s Algorithm.

■ **Table 3** Initial Hints.

Hint Type	Description
Overall Feedback	Resource Leakage Found. The submitted code has some type of resource leakage present.
Basic Hint	CWE-772 : The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
First Hint	You must guarantee that the stream is being closed in every possible scenario.

However, if the prior history is not empty, the intersection between the current set of CWE’s and the ones in the prior history is computed. If the intersection is an empty set, the selected CWE will be computed based on the highest-ranked CWE out of all the reported CWEs. If the intersection is not empty, the selected CWE is computed based on the highest-ranked CWE in the intersection list.

Finally, the hint is then generated based on the selected CWE through a laddering technique. More details on the laddering technique are given in [11, 19].

Table 3 shows the initial hints developed for the initial platform. Each time the player submits the code the platform will release the Overall Feedback and a Hint. This feedback message is a generalized view of the issue found. The Hint given starts with the associated description of the reported CWE and evolves to more specific hints over time. In our instance of the security exercise, the basic hint provides a description of the vulnerability CWE-772 [5]. In the next hint level, a directed message on how to solve the exercise if provided to the player.

In our project, we took the decision to consider security issues with higher priority than functional issues. This means that the generated hints are firstly concerned with security issues and, if no security issues are present, they are concerned with code functionality.

The whole process of analysing submitted code and generating the hints, executed each time a player submits code, usually takes no more than one or two seconds. These time measurements were acquired using a laptop with the following specifications: 12GB of RAM with an Intel Core i5-6200U CPU, running Linux xubuntu version 20.04.

3.2 Empirical Study

We conducted an empirical study using the developed prototype to analyze how the players feel about the platform as a cybersecurity learning system. The test was conducted from 22 to 27 February 2021 through individual online meetings, 11 in total. Each meeting lasted 30 minutes, 20 of those for the solving of the challenge and 10 for a brief interview with the participants. Eleven persons participated in our experiment, with ages ranging from 20 to 35. Nine of the participants are working in the industry, and the remaining two were students in the IT/software development fields. Industry participants are from Germany and Portugal and are software developers for critical infrastructures with more than five years of experience. The academia students were last-year undergraduates from Portugal, studying computer science.

The players received one Java challenge with a security vulnerability to solve. The individual meetings were carried out with each individual participant. While not embedded in a CyberSecurity Challenges event, the present work is used to inform our approach on the implementation of secure coding exercises for the Java programming language.

■ Listing 1 ResourceLeak.java.

```
import java.io.*;

public class ResourceLeak {
    public void writeToFile(File file, String msg) throws IOException {
        FileOutputStream fos = new FileOutputStream(file);
        fos.write(msg.getBytes());
    }
}
```

Listing 1 shows the source code of the exercise presented to the participants during our study. This code was based on the FIO04-J rule [23] of the SEI-CERT coding standards, and contains a resource leakage vulnerability. This type of security issue arises when an opened file handle is not closed. Although Java is a garbage-collected language, the garbage collection mechanism does not work for file handles and database connections. Once these resources are no longer needed they should be immediately closed. A failure to do so can lead to resource starvation of the program or in critical cases resource exhaustion, ultimately leading to a denial-of-service attack [4]. To solve this exercise the participants need to close the file handle either by using a *try-catch-finally* Java block or by implementing a *try with Resources*. In the *try with Resources* scenario the FB Infer tool (V1.0.0) suffers from a False Positive that flags a resource leakage in the code. This problem can be initially addressed in two ways. The first is to wait for the tool to receive an update that would resolve this issue. The second, and chosen course of action, was to give the flag of the challenge on the off chance that the players would encounter this scenario. The participants tested and solved the Java exercise, and experimented with the platform without any restrictions. After solving the challenge, they were asked to complete a small survey containing questions related to their experience, and additional open discussions were held. The survey questions are detailed in table 4.

Answers to the questions are based on a 5-point Likert scale from 1-strongly disagree to 5-strongly agree. Answers were collected anonymously, the participants were instructed about the goal of the research being carried, and they agreed to take part in the study.

■ **Table 4** Survey Questionnaire.

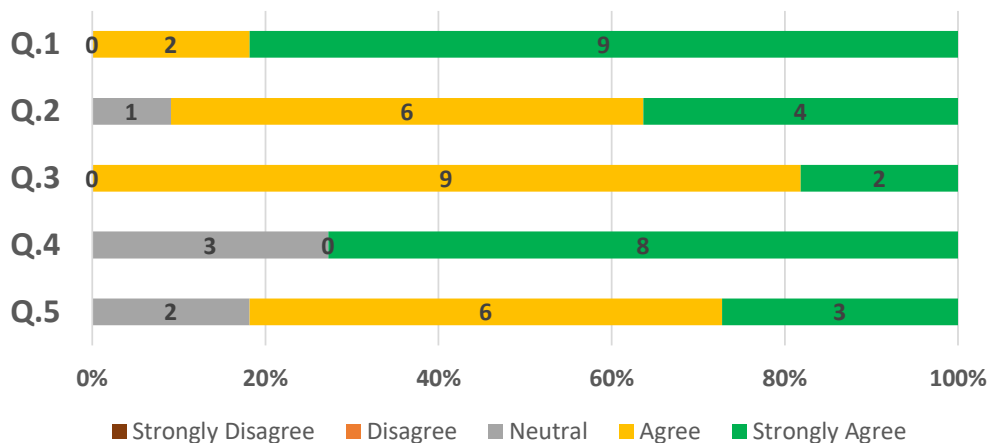
Identifier	Question
Q.1	The error messages and hints issued by the platform are relevant to the exercise.
Q.2	I can relate the hints to the code I have written.
Q.3	The hints provided by the platform make sense to me.
Q.4	If I am given the opportunity, I would like to play more exercises.
Q.5	The hints helped me to understand the problem with the code I have written.

4 Evaluation

In this section, we present the results of our empirical study. Moreover, a critical discussion of the results is also presented. Additionally, we discuss possible threats to the validity of our conclusions. The raw results of our survey can be found in Zenodo [14].

4.1 Results and Discussions

The present work answers the first research question through the presented methodology of using the CyberSecurity Challenges platform to implement Java exercises. In particular, figure 1 shows the three-stage method that we have implemented to automatically assess the level of security of the Java code from the players to the game. To answer the second research question, we have evaluated a number of openly available tools. These results are summarized in table 2. In the following, we present a discussion related to the third research question, on how the players perceive the platform as a means to learn secure coding in the Java programming language.



■ **Figure 3** Results of Empirical Study in the Industry.

Figure 3 shows the results of our empirical study, in relation to the survey questions (see table 4). Our preliminary results show that all participants either agreed or strongly agreed that the platform’s hints make sense to them (Q.3) and are related to the programming exercise (Q.1). The majority of the participants agree or strongly agree that the hints are related to the code they wrote (Q.2), with only one participant giving a neutral answer. More than 80% of the participants agree or strongly agree that the platform’s hints help to understand the security issues present in the code (Q.5). About 72% of the participants strongly agree that, given the opportunity, they would like to play additional exercises (Q.4); however, three participants (27%) are not sure.

Although Q.2, Q.4, and Q.5 show mostly positive results, one, three, and two neutral answers were obtained for each question, respectively. Further studies are needed to understand possible reasons that led the participants to give these answers. Overall, the outcome obtained in this preliminary study, both through the survey and additional discussions, indicates that the participants welcome the exercises. Furthermore, the hints generated by our proposed automatic security assessment method, which uses widely available open-source tools, received positive ratings from the participants of our study. This is in line with previous similar studies for other programming languages, and gives the authors encouragement to further develop the platform.

Although most participants come from the industry, we hypothesize that similar results might be obtained for students in academia. Furthermore, we also hypothesize that the same method can raise awareness on other coding issues, e.g., code-smells, naming conventions, and code format.

This paper is focused on the Java programming language and the security vulnerabilities associated with it. However, the method to automatically assess the submitted code can be applicable and extendable to any programming language. Just, for each programming language the set of vulnerabilities is different and thus the assessment method needs to be updated accordingly. Although these tests are being developed for the industry, they can also be applied in academia. The developed artifact can be incorporated in programming courses (mainly junior-level) to teach these concepts to students, possibly leading to better prepared and security-aware software developers. The purpose of the artifact itself can be shifted. Instead of focusing 100% on the security part of code development, it can be enhanced to deal with and cover other issues, e.g., code-smells or naming convention.

4.2 Threats to Validity

Eleven participants took part in our preliminary empirical study, which is in line with similar preliminary empirical studies in industrial settings. The results were collected anonymously, and the gathered data does not allow us to understand the difference between participants from industry and students. Nevertheless, since the overall result is positive, we believe that an eventual bias does not affect our conclusions. Furthermore, our results are in alignment with previous studies. Static-application security tools are known to produce false-positives and false-negatives. To counteract these issues, we use the following strategy: (1) our platform filters the findings based on our tools' experience, (2) we use additional security-tests to cover false-negatives, and (3) we test the exercise extensively before deployment.

5 Conclusions and Further Work

Over the past decade, the number of cybersecurity incidents has been increasing. To counteract this issue, companies can follow several strategies to reduce the number of vulnerabilities in their products and services. These strategies aim to increase resilience to malicious attacks. In this work, we focus on the human-factor, through awareness training. To achieve this, we adapt and extend previous work on CyberSecurity Challenges to automatically perform security assessment of Java challenges and generate hints for players. This paper gives a brief overview of how this automatic assessment can be performed using openly available tools enabling practitioners to reproduce our results. We also introduce a hint-generation algorithm that is used to interact with the player. Finally, we perform an empirical evaluation of our proposed method together with eleven participants. Nine participants are working in the industry, and two are students in the software development

field. Our preliminary results show that the participants can understand the hints generated through our method and agree that these hints are helpful to solve the Java challenge. In further work, the authors would like to implement additional Java challenges and perform a detailed empirical evaluation of the artifact in an industry setting. In this future work, we would like to understand how the hint system targeting Java challenges can be improved, in particular addressing to which extent and which strategies can cope with the fact that findings from static-application security testing tools can potentially contain false-positives and false-negatives.

References

- 1 Uğur Bakan and Ufuk Bakan. Game-Based Learning Studies in Education Journals: A Systematic Review of Recent Trends. *Atualidades Pedagógicas*, pages 119–145, July 2018. doi:10.19052/ap.5245.
- 2 Marílio Cardoso, António Vieira de Castro, Álvaro Rocha, Emanuel Silva, and Jorge Mendonça. Use of Automatic Code Assessment Tools in the Programming Teaching Process. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASICs)*, pages 4:1–4:10, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASICs.ICPEC.2020.4.
- 3 MITRE Corporation. Common Weakness Enumeration. Online, Accessed 4 July 2019. URL: <https://cwe.mitre.org/>.
- 4 MITRE Corporation. Common Weakness Enumeration - 404. Online, Accessed 4 July 2019. URL: <https://cwe.mitre.org/data/definitions/404.html>.
- 5 MITRE Corporation. Common Weakness Enumeration - 772. Online, Accessed 4 July 2019. URL: <https://cwe.mitre.org/data/definitions/772.html>.
- 6 Ian Cullinane, Catherine Huang, Thomas Sharkey, and Shamsi Moussavi. Cyber Security Education Through Gaming Cybersecurity Games Can Be Interactive, Fun, Educational and Engaging. *J. Computing Sciences in Colleges*, 30(6):75–81, June 2015.
- 7 Department of Homeland Security, US-CERT. Software Assurance. Online, Accessed 27 September 2020. URL: <https://tinyurl.com/y6pr9v42>.
- 8 Ralph Dörner, Stefan Göbel, Wolfgang Effelsberg, and Josef Wiemeyer. *Serious Games: Foundations, Concepts and Practice*. Springer International Publishing, 1 edition, 2016. doi:10.1007/978-3-319-40612-1.
- 9 Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security. In *IEEE Symposium on Security and Privacy*, pages 121–136, San Jose, CA, USA, 2017. IEEE Computer Society. doi:10.1109/SP.2017.31.
- 10 Tiago Gasiba, Kristian Beckers, Santiago Suppan, and Filip Rezabek. On the requirements for serious games geared towards software developers in the industry. In Daniela E. Damian, Anna Perini, and Seok-Won Lee, editors, *27th IEEE International Requirements Engineering Conference, RE 2019, Jeju Island, Korea (South), September 23-27, 2019*. IEEE, 2019. URL: <https://ieeexplore.ieee.org/xpl/conhome/8910334/proceeding>.
- 11 Tiago Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Sifu - a cybersecurity awareness platform with challenge assessment and intelligent coach. In *Special Issue of Cyber-Physical System Security of the Cybersecurity Journal*, Online, October 2020. SpringerOpen.
- 12 Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. What happens when software developers are (un)happy. *Journal of Systems and Software*, 140:32–47, June 2018. doi:10.1016/j.jss.2018.02.041.
- 13 Norman Hansch and Zinaida Benenson. Specifying IT Security Awareness. In *25th International Workshop on Database and Expert Systems Applications, Munich, Germany*, pages 326–330, September 2014. doi:10.1109/DEXA.2014.71.

- 14 Luis Afonso Casqueiro. Automated Java Challenges' Security Assessment for Training in Industry – Preliminary Results. *Zenodo*, February 2021. . doi:10.5281/zenodo.4740829.
- 15 Na Meng, Stefan Nagy, Danfeng Daphne Yao, Wenjie Zhuang, and Gustavo Arango. Secure Coding Practices in Java: Challenges and Vulnerabilities. In *IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 372–383, May 2018. doi:10.1145/3180155.3180201.
- 16 Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A DeLong, Justin Cappos, and Yuriy Brun. API Blindspots: Why Experienced Developers Write Vulnerable Code. *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 315–328, August 2018. (USENIX) Association, Baltimore, MD, USA, ISBN: 978-1-939133-10-6. URL: <https://www.usenix.org/conference/soups2018/presentation/oliveira>.
- 17 Suri Patel. 2019 Global Developer Report: DevSecOps Finds Security Roadblocks Divide Teams. Online, Accessed 18 July 2020. URL: <https://tinyurl.com/3z57t32d>.
- 18 Alison DeNisco Rayome. The 3 Least Secure Programming Languages, March 2019. URL: <https://www.techrepublic.com/article/the-3-least-secure-programming-languages/>.
- 19 Tim Rietz and Alexander Maedche. LadderBot: A Requirements Self-Elicitation System. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 357–362. IEEE, 2019.
- 20 Marc Schönefeld. *Java-Security: Sicherheitslücken identifizieren und vermeiden*. MITP-Verlags GmbH & Co. KG, 2011.
- 21 Maung Sein, Ola Henfridsson, Sandeep Puro, Matti Rossi, and Rikard Lindgren. Action Design Research. *MIS Quarterly*, 35(1):37–56, March 2011. doi:10.2307/23043488.
- 22 Software Engineering Institute, Carnegie Mellon. SEI CERT Oracle Coding Standard for Java. Online, Accessed 11 June 2018. URL: <https://tinyurl.com/ypm4mnj8>.
- 23 Software Engineering Institute, Carnegie Mellon. SEI CERT Oracle Coding Standard for Java - FIO04-J. Release resources when they are no longer needed. Online, Accessed 11 June 2018. URL: <https://wiki.sei.cmu.edu/confluence/display/java/FIO04-J.+Release+resources+when+they+are+no+longer+needed>.
- 24 Silvio Sorace, Elisabeth Quercia, Ernesto La Mattina, Charalampos Z. Patrikakis, Liz Bacon, Georgios Loukas, and Lachlan Mackinnon. *Serious Games: An Attractive Approach to Improve Awareness*, pages 1–9. Springer International Publishing, Springer Cham, 2018.