


# Code Review for CyberSecurity in the Industry: Insights from Gameplay Analytics

Andrei-Cristian Iosif ✉ 


Universität der Bundeswehr München, Germany  
Siemens AG, München, Germany

Ulrike Lechner ✉ 

Universität der Bundeswehr München, Germany

Maria Pinto-Albuquerque ✉ 

Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Tiago Espinha Gasiba ✉ 

Siemens AG, München, Germany

---

## Abstract

In pursuing a secure software development lifecycle, industrial developers employ a combination of automated and manual techniques to mitigate vulnerabilities in source code. Among manual techniques, code review is a promising approach, with growing interest within the industry around it. However, the effectiveness of code reviews for security purposes relies on developers' empowerment and awareness, particularly in the domain-specific knowledge required for identifying security issues. Our study explores the use of *DuckDebugger*, a serious game designed specifically to enhance industrial practitioners' security knowledge for code reviews. By exploring analytics data collected from game interactions, we provide insights into player behavior and explore how the game influences their approach to security-focused code reviews. Altogether, we explore data from 13 events conducted in the industry together with 224 practitioners, and derive metrics such as the time it takes participants spend to reviewing a line of code and the time required to compose a comment. We offer empirical indicators on how serious games may effectively be utilized to empower developers, propose potential design improvements for educational tools, and discuss broader implications for the use of Serious Games in industrial settings. Furthermore, our discussion extends to include a discussion outlining the next steps for our work, together with possible limitations.

**2012 ACM Subject Classification** Security and privacy → Software and application security; Applied computing → Collaborative learning; Applied computing → E-learning; Security and privacy → Software security engineering

**Keywords and phrases** Cybersecurity, Code Review, Developer Empowerment

**Digital Object Identifier** 10.4230/OASICS.ICPEC.2024.14

**Funding** This work is partially financed by Portuguese national funds through FCT – Fundação para a Ciência e Tecnologia, I.P., under the projects FCT UIDB/04466/2020 and FCT UIDP/04466/2020. Furthermore, the third author thanks the Instituto Universitário de Lisboa and ISTAR, for their support. We acknowledge funding for project LIONS by dtec.bw. Andrei-Cristian Iosif and Tiago Gasiba acknowledge the funding provided by the Bundesministerium für Bildung und Forschung (BMBF) for the project CONTAIN (FKZ 13N16585).

## 1 Introduction

In recent years, the cybersecurity community has witnessed how purposefully introduced software vulnerabilities can be weaponized into sophisticated supply chain attacks. One such example of this is the Lazarus group [10] Advanced Persistent Threats (APT). Such attacks abuse trust relationships and result in malicious code being injected into legitimate software.



© Andrei-Cristian Iosif, Ulrike Lechner, Maria Pinto-Albuquerque, and Tiago Espinha Gasiba; licensed under Creative Commons License CC-BY 4.0

5th International Computer Programming Education Conference (ICPEC 2024).

Editors: André L. Santos and Maria Pinto-Albuquerque; Article No. 14; pp. 14:1–14:11

OpenAccess Series in Informatics



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

One way to combat this is through rigorous code review processes. Reviews can serve as an additional security measure against targeted breaches, and also catch less malignant security issues from legitimate developers, that might otherwise be accidentally overlooked.

Industrial software systems, particularly those that manage critical infrastructure, are an area of special concern when accounting for cybersecurity - with stakes including economic concerns and public safety. Consequently, such systems must adhere to stringent standards and compliance regulations, such as the IEC 62433 [7].

One way to achieve security and compliance in an industrial environment is a bottom-up approach that begins with empowering developers. Training programs focused on security can significantly enhance developers' ability to detect and mitigate vulnerabilities earlier in the development process.

One innovative tool in this effort is DuckDebugger, a serious game (SG) designed specifically to teach software developers to perform code reviews with a focus on cybersecurity. We use SGs as a medium for information delivery, as the authors possess extensive knowledge in this area. The proposed game offers an environment where developers are asked to review snippets of vulnerable code, and are able to consult the output from security tools. Through engaging with our game, developers can hone their skills in a practical and realistic environment.

This paper presents the results from analyzing gameplay data from events conducted in industrial contexts. Our objective is to understand the interactions taking place in the DuckDebugger. By exploring player behavior, our analysis takes a first step towards measuring how players engage in code review, in order to gauge how the game sizes towards empowering practitioners. The insights from our study show the potential for serious games to improve security training in software development. Furthermore, we identify design implications for our game and optimization of the game experience for the participants.

The paper is organized as follows: Section 2 will introduce related work relevant to our study. In Section 3, we present our work's context and showcase the game artifact's relevant aspects. Section 4 presents our findings, exploring their implications and discussing potential limitations. Lastly, Section 5 lays out the conclusions of our work, together with discussing further steps.

## **2 Related Work**

This section introduces related work in the two main areas of interest under which our game falls, namely cybersecurity training through serious games, and code review.

### **2.1 Serious Games for Cybersecurity Education**

A serious game is defined by Susi et al. as “a game designed with a primary objective other than pure entertainment” [18]. Furthermore, indicated they are effective in disseminating takeaways to their users, as shown in an experience report of Namin et al [17] and through a literature review conducted by Hendrix et al [4].

Roepke et al. provide a comprehensive overview of SGs focused on cybersecurity, and explore how many games available for end-users without prior knowledge exist, as well as whether they teach sustainable knowledge and skills [15]. Their findings show that although there is a growing number of SGs for cybersecurity, the content they target is often lacking relevance, focusing rather on factual knowledge without context.

Capture The Flag (CTF) competitions, a popular form of competitive serious games in cybersecurity, have been extensively studied for their educational value. Research by Culliane et al. [2] and Švábenský et al. [20] provide an overview of CTFs seen through the educational angle.

Unlike traditional CTFs and challenges thereof, which often emphasize offensive practices, the game discussed in this work focuses on defense and patching.

Previous work done by the authors on this topic explores the requirements of defensive cybersecurity SGs [3], defining 15 challenge requirements suitable for industrial requirements. The game is developed using an iterative design approach based on Design Science Research (DSR) principles, as outlined by Sein [16]. In this second iteration, we have incorporated feedback from participants, specific to DSR guidelines, and have successfully met 11 out of the 15 requirements. We plan to address the remaining requirements, should the participants deem it necessary.

Švábenský et al. [19] review how cybersecurity training data can improve educational research. Their work offers an overview on how training data can to understand and support cybersecurity learning. However, their approach notes that it only targets academic settings and exercises with an offensive focus.

## 2.2 Industrial Code Review

Regarding standardization within the industry, Moyon et al. [13] delve into the practical challenges related to integrating security into an agile software development and explore how the requirement for industrial compliance shapes this process.

The IEC 62443 series of standards was developed to cover the security of industrial automation and control systems throughout their lifecycle. Relevant to our work, the IEC 62443-4-1 [7] and IEC 62443-4-2 [8] standards underscore how code review in the development lifecycle is important for cybersecurity, and the emphasize the need for reviewers to possess specialized knowledge on performing secure code review.

Other standards of interest to our work are: ISO/IEC 20246 [6], which provides a generic framework for work product reviews applicable across various organizational roles, and the ISO/IEC TR 24772-1 standard [9], which offers programming-language agnostic guidance on avoiding coding vulnerabilities.

We seek to address the most prevalent vulnerabilities encountered in software by introducing vulnerabilities. To this end, we draw from the the CWE Top25 [12] and OWASP Top10 [14], as these resources are well-known standards, with industry-wide acceptance.

MacLeod et al. show that code reviews often do not find critical bugs that would block a code submission, instead highlighting issues related to long-term code maintainability [11]. They argue that effective code reviews require specific skills and that the social dynamics within teams significantly influence the reviewing process. They suggest that the current practices in code reviews are often inefficient and call for a more sophisticated approach to integrating code reviews into software engineering workflows. Building on these findings, our work seeks to address these deficiencies by employing the DuckDebugger as a means to enhance practitioners' skills necessary for effective code review.

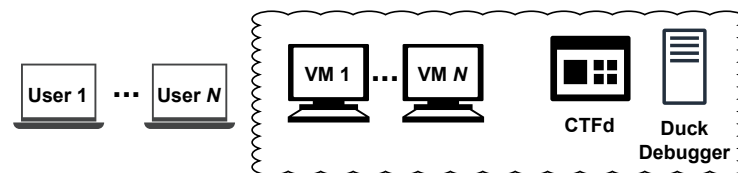
Bosu et al. analyze the effectiveness of code review comments in Microsoft projects and find that useful feedback improves both code quality and developers' skills [1]. They show that the usefulness of comments tends to increase with the reviewer's experience but is negatively correlated with the size of the review. The study provides recommendations for increasing the effectiveness of code reviews, such as optimizing the number of files in a review and enhancing reviewer experience through targeted training. We integrate their findings in the design of the game, which we use for training industrial developers.

### 3 Methodology

In this section, we describe the methodology behind the collected data. First, we present the industrial context in which the DuckDebugger is deployed. We proceed by introducing the game artifact itself, highlighting the important elements of our data.

#### 3.1 Context

Practitioners are invited to try out our serious game as part of a workshop on developer empowerment on the topic of cybersecurity. Such workshops consist of two parts: first, a classroom-style presentation of common attacks and mitigations (1-2 days, depending on the event), followed by a full-day CTF-style event. On the last day, participants organize themselves in teams, where they compete against each-other by solving various types of security-related challenges. One of the types are the code review challenges discussed in this work. Although we integrate the delivery of our game in a CTF setting for our events, this is not strictly necessary, as the game is designed to be a self-standing application.



■ **Figure 1** Architecture.

Figure 1 presents an overview of the infrastructure behind the hands-on part of the event. Participants only require a laptop, with which they can connect to a fully provisioned virtual machine for all the available challenges. Upon solving a challenge, players redeem points for their team in a dashboard (CTFd). We utilize the dashboard mechanic of the CTF genre to foster competitiveness among participants.

When the event concludes, the winning team is announced and congratulated. This is followed by a gathering feedback from the participants (through surveys and semi-structured interviews), and a wrap-up session which discusses challenges which the players found difficult.

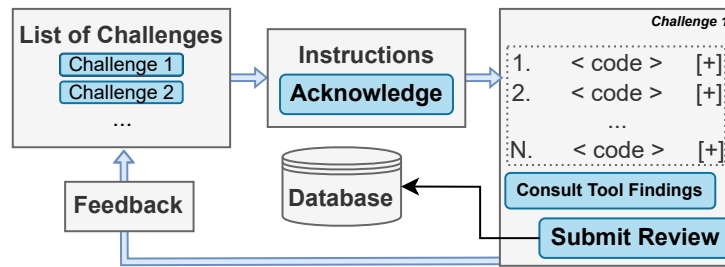
At the time of writing, the DuckDebugger has been trialed across 13 industrial events, with a total of 224 participants. An overview of these events is presented in Table 1.

■ **Table 1** Events Overview.

Event Number	1	2	3	4	5	6	7	8	9	10	11	12	13
Date	05.2023	05.2023	06.2023	11.2023	11.2023	11.2023	12.2023	12.2023	01.2024	01.2024	02.2024	02.2024	02.2024
Place	Online	Online	United Kingdom	Germany (Hybrid)	Germany	China	Online	China	Online	Germany	Germany	Online	Germany (Hybrid)
Number of Participants (Total: 224)	14	7	16	9	20	24	30	20	16	15	24	17	12

#### 3.2 Game Artifact

The focus of this work, the DuckDebugger platform, is a self-standing web application and is hosted as a separate server in the same cloud environment as the participants' virtual machines, in an AWS Virtual Private Cloud. This design choice was taken to ensure that the game can offer flexibility in how it can be delivered.



■ **Figure 2** Artifact Overview.

Figure 2 introduces an interaction diagram for the game: The user starts by viewing a welcome screen with platform instructions. After this, they choose from various code review challenges categorized by programming language and application type, such as web or embedded programming. They then receive a source code snippet with vulnerabilities and poor practices, which they are asked to annotate with comments about security findings.

At the time of writing, participants can choose from 28 exercises that cover 4 programming languages: Java, C#, Python, and JavaScript. The vulnerabilities in our game’s snippets include common security malpractices, from the CWE Top25 [12] and OWASP Top10 [14].

Figure 3 shows the main game interface, organized in tabular form, with columns for user comments, the code under review, and the intended solution (to be displayed after a user successfully solves a challenge). Users input their comments and can reference SAST tool findings alongside the code. The integration of SAST aims to enhance users’ skills with real-world tools and educate them on recognizing false positives/negatives.

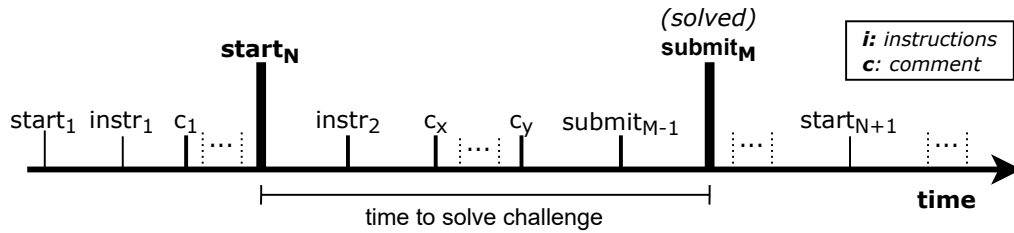


■ **Figure 3** Interface.

Upon submitting their review comments, the DuckDebugger evaluates how many vulnerabilities a player has found, and records the interactions with the platform into a database for later analysis. If a player identifies at least 50% of the vulnerabilities, they are presented the full solution as feedback. This benchmark is based on prior research from the authors [5], where it was found that trainees identify, on average, about half of the vulnerabilities detected by experts.

## 4 Results and Discussion

To help understand the player behavior analysis, we consider the following: Players have the autonomy to engage with, resolve, or abandon any challenge at will, in any preferred order.



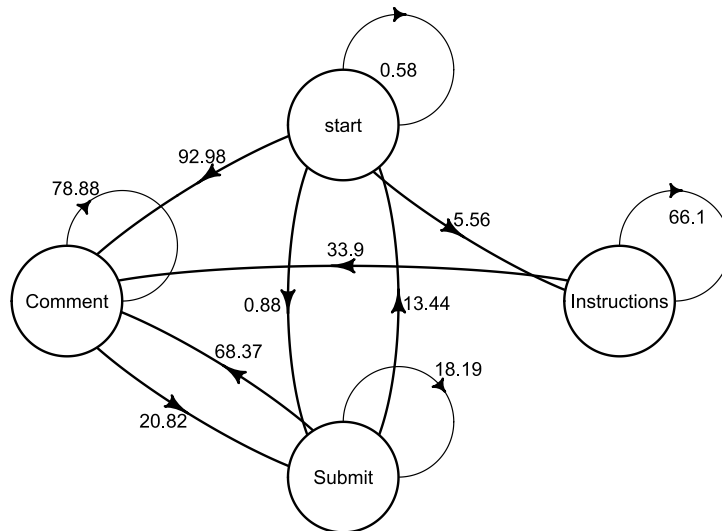
■ **Figure 4** Timeline of interactions for a (user,challenge) pair.

An example timeline of interactions is presented in Figure 4, illustrating a *single user's* interaction sequence for a *single challenge*. Possible interactions for a user are: starting the challenge (**start**), consulting the instructions (**i**), writing a comment (**c**), and submitting their solution attempt (**submit**).

The figure has been simplified for clarity – the collected data covers the multiple users' interactions, for multiple challenges, where all entries of users and challenges are chronologically interlaced.

The time it took a given user to solve a specific challenge is computed as the timestamp difference between the first occurrence of a **submit** interaction marked as *solved* by the platform, and the first **start** interaction which precedes it.

### 4.1 Player Behavior Model



■ **Figure 5** Platform Interactions: Player behavior.

Based on the collected players' interactions, we can construct a transition matrix between possible actions in the game. Figure 5 models the player behavior and highlights the transition probabilities between actions. Most notably in this figure, we can observe that:

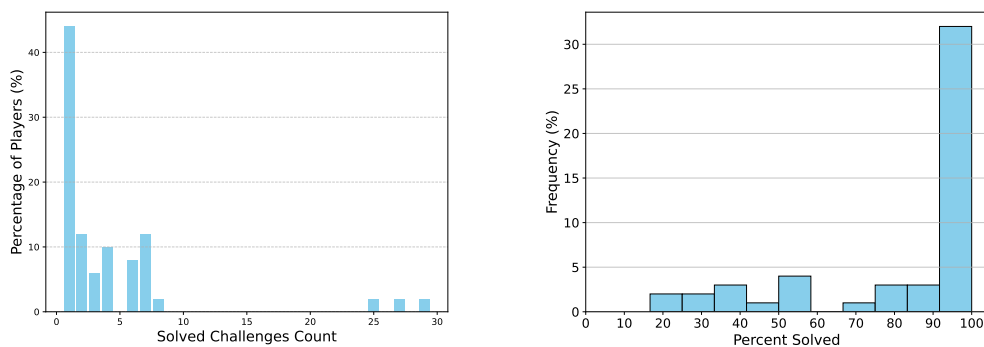
- 66,1% of players re-check the instructions. This could likely be due to habitual interactions with similar interface elements, suggesting a conditioned response to dismiss such overlays without thorough engagement. Empirical surveys, however, indicate that the majority find the instructions clear. Additionally, the absence of a **submit** → **instructions** transition indicates that the task is well understood.

Similarly, the **start** → **instructions** transition occurs because users are accustomed to dismissing pop-ups quickly. This would suggest a need for different UI choices, such as waiting for a timer to expire before being able to click away the instructions pop-up.

- 18.19% of users press the submit button repeatedly (**submit** → **submit**). This would indicate a need for a visual cue to confirm that the button has been activated successfully.
- **start** → **start** interactions account for 0.58% of interactions, possibly stemming from accidental page refreshes.
- One in five comments is immediately followed by a submission attempt – 20.82% on the **comment** → **submit** transition. If this percentage were lower, it would suggest that users are overthinking their submissions. Conversely, a higher percentage might indicate players trying to exploit the system for a competitive edge by rapidly resubmitting their comments.

## 4.2 Solved Challenges Counts and Percentages

Another perspective on player behavior involves examining the number of challenges each player successfully completes. It is useful to observe whether players abandon challenges before finishing them.



(a) Count Solved.

(b) Percent Solved.

■ **Figure 6** Metrics: Solving Challenges.

Figure 6a illustrates that the majority of participants solve fewer than 10 challenges, with a noticeable peak at just one challenge. This trend may be attributed to the specific conditions of the event, such as time constraints and the diversity of available challenges.

Figure 6b shows that more than 30% of players solve all challenges they start, and more than 50% solve half or more. This demonstrates a commendable level of persistence among the players, indicating that they generally complete the challenges they begin.

## 4.3 Time to solve a challenge

Based on the time it takes players to solve a challenge, we derive and examine the duration participants spend reviewing a line of code (LoC) and the time required to compose a comment. As challenges have code snippets of varying length, the times are normalized to each challenge's LoC. We group our data by programming language, average it, and present our findings in Table 2.

■ **Table 2** Times.

Programming Language	Avg. time to review one Line of Code (s.)	Avg. time to write one comment (s.)
csharp	12.56	4.87
go	3.44	1.94
java	1.57	1.67
javascript	3.47	1.95
python	12.57	9.32

In the case of Java, the lower numbers can be attributed to the language’s syntactic noise (*boilerplate code*), which may result in a lower time per LoC. Additional influencing factors include the diverse backgrounds of the participants (i.e. few proficient Python programmers across all events), which could also play a significant role in the observed results.

The correlation coefficient between the two metrics presented in Table 2 is 0,873. This indicates that the time participants take to comment is relatively consistent across different programming languages. Such strong correlation could indicate that the measured times are an indicator of the players’ proficiency levels with the programming languages, rather than an indicator of review times being dependent on the programming language. Nonetheless, further studies would be required to establish a definitive conclusion in this direction.

Furthermore, we can explore how the participants’ solving times evolve between consecutive challenges. Table 3 presents the median time it takes a player to solve a challenge. We truncate our findings at 7 challenges, as Figure 6a shows that few players only solve more than 7 challenges, which would challenge the statistical relevance of these findings. Although our data is truncated, preliminary findings indicate a non-linear improvement pattern.

■ **Table 3** Time to solve a challenge.

Number of Solved Challenges	1	2	3	4	6	7
Median Time To Solve (s.) (normalized to challenge LoC)	7.72	18.87	3.29	7.55	6.48	3.14

We can observe that there is no steady increase or decrease in solving time, between the number of challenges a players solves. Based on our experience in designing serious games, this non-linear time pattern could point to multiple insights and/or reflect our design choices: the challenges have variable difficulty, and participants have diverse problem solving approaches as they learn and adapt to the platform. Had there been a steady increase, this would indicate that the challenges result in tiredness or a plateau in learning efficiency. Conversely, the lack of a steady decrease in the solve time indicates that the DuckDebugger has a varied repertoire of code review challenges, where players cannot *game the system* through recalling the solutions to the challenges they previously encountered and solved.

#### 4.4 Knowledge Exchange

The nature of the event has players organized in teams which compete for points across multiple challenge categories, some of which are code review challenges delivered through the DuckDebugger. We observed during the events that most, if not all teams competitively optimize their strategy by splitting up across challenge types. With proper intra-team coordination, this would translate into one player per team solving the DuckDebugger challenges at any given time.



Nonetheless, while conducting the events, we observed players of the same teams often pause to share newfound knowledge from the challenges. We back this statement through an observation in our aggregated data: 28.2% of players revisit a challenge after having solved it. This indicates to us that more than a quarter of the participants likely exchange the gained knowledge with their team-mates, to share newly gained information.

Since the dashboard interface indicates to the other players of a team that a challenge is already solved, this would rule out most accidental revisits of a challenge. Furthermore, there is no competitive advantage in solving a challenge again.

This finding about players' knowledge exchange underlines the relevance of code review as an industrial practice fostering developer empowerment, as it shows an organic tendency towards knowledge sharing even under a competitive setting. We thus reinforce the importance of collaborative learning and the dissemination of best practices across the workforce, and show that code review is a good vector towards achieving this. This observation reinforces the value of code review not only as a skill but also as a catalyst for fostering developer empowerment and the spread of best practices across the industry.

The competitive yet collaborative environment created by the DuckDebugger game illustrates how serious games can bridge the gap between individual learning and team-based knowledge dissemination. This aspect is crucial in real-world applications where cybersecurity is a collective responsibility. By integrating these insights into future game iterations, DuckDebugger can further enhance its impact on cybersecurity training.

## 4.5 Discussion on Validity and Limitations

The study on the use of DuckDebugger for code review training in cybersecurity presents potential threats to validity, particularly in the context of generalization and external validity.

First, the events are conducted in a controlled setting, which may not accurately reflect developers' typical working environment. This difference might influence the behavior and performance of participants, as developers in a competitive, time-constrained event might interact with the game differently compared to a regular work setting.

The analysis of gameplay data might overlook deeper cognitive and learning processes involved in vulnerability identification and mitigation, focusing primarily on observable metrics such as time spent and challenge completion rates. Furthermore, addressing the long-term impact of the game is unfeasible in an industrial setting, as developer teams and individual responsibilities shift with time. Learning trends, solving efficiency and proficiency can thus only be observed within the scope of individual events. Nevertheless, preliminary findings show neither a learning plateau nor a saturation of gained knowledge.

Furthermore, the event's structure, which organizes individual players into teams, likely impacted data collection in terms of sample size. We observed during our the moderation of our event that users within the same team tend to split between challenge types, to gain more points in total. Due to how data is anonymized, we cannot present a percentage of players and teams that choose to follow this strategy. As we observed players dividing their efforts across different challenge types to optimize team points, this typically resulted in not all people from a team addressing the review challenges. Nonetheless, as the data is collected at an individual level, and not aggregated by teams, individual performance metrics are accurately represented, independent of team dynamics.

As a general note, although our game has been embedded in a CTF setting, though this is not strictly necessary. The implementation of the game makes it independent of a CTF setting. We believe our findings to be similar, had the game been introduced under a different structure.

Nonetheless, our conclusions align with previous industrial research around serious games. This, coupled with the typical limitations inherent to design science studies carried out in the industry, leads us to consider that our findings should not significantly diverge, had the event been restructured to optimize for data collection instead of learning outcome.

## 5 Conclusions

This study explores the DuckDebugger game as a tool for training developers in cybersecurity-focused code review through gameplay analytics. Our results extrapolate a player behavior model which can be used by practitioners to design similar games, and explore metrics related to players solving challenges. Notably, the game design, which incorporates elements of competition and immediate feedback, fosters engagement and learning among participants.

Notably, the finding that over a quarter of participants revisit challenges even after solving them highlights an organic tendency towards knowledge sharing within teams, emphasizing the collaborative nature of learning, even in competitive settings.

Our findings suggest that serious games like DuckDebugger can enhance cybersecurity education among industrial developers. This aligns with the results of previous related work, reinforcing the importance of tailored training tools in improving cybersecurity skills (i.e. code review) and fostering collaborative learning environments. The interactive and practical nature of our game provides a hands-on learning environment.

Our game contributes to academic research by exploring the use of use of SGs in an industrial setting, specifically centered around empowering developers through code review for cybersecurity. We follow DSR principles and focus on a *defensive* approach to disseminating cybersecurity knowledge, by targeting mitigation techniques in our game, instead of exploitation of vulnerabilities.

Future work aims to address the identified refinement needs, including game's design based on the collected feedback and interaction analytics. Additionally, we plan to test additional scenarios and further evaluate the utility of the game as perceived by the participants.

---

## References

- 1 Amiangshu Bosu, Michaela Greiler, and Christian Bird. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 146–156, Florence, Italy, 2015. IEEE. doi:10.1109/MSR.2015.21.
- 2 Ian Cullinane, Catherine Huang, Thomas Sharkey, and Shamsi Moussavi. Cyber security education through gaming cybersecurity games can be interactive, fun, educational and engaging. *J. Comput. Sci. Coll.*, 30(6):75–81, June 2015.
- 3 Tiago Espinha Gasiba, Kristian Beckers, Santiago Suppan, and Filip Rezabek. On the requirements for serious games geared towards software developers in the industry. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 286–296, 2019. doi:10.1109/RE.2019.00038.
- 4 Maurice Hendrix, Ali Al-Sherbaz, and Victoria Bloom. Game based cyber security training: are serious games suitable for cyber security training? *International Journal of Serious Games*, 3(1), March 2016. doi:10.17083/ijsg.v3i1.107.
- 5 Andrei-Cristian Iosif, Tiago Espinha Gasiba, Ulrike Lechner, and Maria-Pinto Albuquerque. Raising awareness in the industry on secure code review practices. In *CYBER 2023: The Eighth International Conference on Cyber-Technologies and Cyber-Systems*, pages 62–68. IARIA, September 2023.
- 6 ISO/IEC 20246:2017. Software and systems engineering – Work product reviews. Standard, International Organization for Standardization, Geneva, CH, 2017.

- 7 ISO/IEC 64223-4-1:2018-1. ISO/IEC 62443-4-1:2018 Security for industrial automation and control systems – Part 4-1: Secure product development lifecycle requirements. Standard, International Organization for Standardization, Geneva, CH, January 2018.
- 8 ISO/IEC 64223-4-2:2019-12. Security for Industrial Automation and Control Systems – Part 4-2: Technical Security Requirements for IACS Components. Standard, International Electrical Commission, Geneva, CH, January 2019. ISBN 978-2-8322-6597-0.
- 9 ISO/IEC TR 24772-1:2019. Programming languages – Guidance to avoiding vulnerabilities in programming languages – Part 1: Language-independent guidance. Standard, International Organization for Standardization, Geneva, CH, 2019.
- 10 Peter Kálnai. Lazarus campaigns and backdoors in 2022-2023. In *Proceedings of the Virus Bulletin International Conference*, London, United Kingdom, October 2023.
- 11 Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. Code reviewing in the trenches: Challenges & best practices. *IEEE Software*, 35(4):34–42, 2017.
- 12 MITRE Corporation. CWE Top 25 Most Dangerous Software Weaknesses. <http://bit.ly/mitre25>, 2023. Online, accessed 2023.07.24.
- 13 Fabiola Moyon, Daniel Mendez, Kristian Beckers, and Sebastian Klepper. How to integrate security compliance requirements with agile software engineering at scale? In Maurizio Morisio, Marco Torchiano, and Andreas Jedlitschka, editors, *Product-Focused Software Process Improvement*, pages 69–87, Cham, 2020. Springer International Publishing.
- 14 OWASP Foundation. OWASP Top10:2021. <https://owasp.org/Top10>, 2021. Online, accessed 2023.07.24.
- 15 Rene Roepke and Ulrik Schroeder. The problem with teaching defence against the dark arts: A review of game-based learning applications and serious games for cyber security education. In *Proceedings of the 11th International Conference on Computer Supported Education*. SCITEPRESS – Science and Technology Publications, 2019. doi:10.5220/0007706100580066.
- 16 Maung K. Sein, Ola Henfridsson, Sandeep Puro, Matti Rossi, and Rikard Lindgren. Action Design Research. *MIS Quarterly*, 35:37–56, 2011.
- 17 Akbar Siami Namin, Zenaida Aguirre-Muñoz, and Keith Jones. Teaching cyber security through competition an experience report about a participatory training workshop. In *7th Annual International Conference on Computer Science Education: Innovation & Technology (CSEIT 2016)*, CSEIT. Global Science & Technology Forum (GSTF), October 2016. doi:10.5176/2251-2195\_cseit16.39.
- 18 Tarja Susi, Mikael Johannesson, and Per Backlund. Serious games: An overview. Technical report, IKI Technical Reports, 2007.
- 19 Valdemar Švábenský, Jan Vykopal, Pavel Čeleda, and Lydia Kraus. Applications of educational data mining and learning analytics on data from cybersecurity training. *Education and Information Technologies*, 27(9):12179–12212, May 2022. doi:10.1007/s10639-022-11093-6.
- 20 Valdemar Švábenský, Pavel Čeleda, Jan Vykopal, and Silvia Brišáková. Cybersecurity knowledge and skills taught in capture the flag challenges. *Computers and Security*, 102:102154, 2021. doi:10.1016/j.cose.2020.102154.