Catching What the Borrow Checker Can't: Towards Serious Game-Based Security Training for Rust Developers

Frederico d'Abreu
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal

Tiago Espinha Gasiba
Siemens AG, Munich, Germany

Sathwik Amburi
Siemens AG, Munich, Germany

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

Abstract

Cybersecurity incidents cause significant financial damage to organizations, making secure software development a critical industry priority. Although Rust offers strong memory safety guarantees, developers can still introduce security vulnerabilities through improper coding practices. In this paper, we present the preliminary design of a serious game for secure Rust coding training in an industrial setting. We investigate real-world Rust vulnerabilities, map (five of) them to CWEs, and propose challenge scenarios alongside a structured evaluation protocol for validation by industry security experts. Our design incorporates defense-oriented coding tasks, a three-level AI-powered hint system, and automated backend evaluation, grounded in established pedagogical principles such as situated learning and cognitive load management. This work provides a foundation for future implementation and empirical evaluation of learning outcomes with professional software developers.

2012 ACM Subject Classification Security and privacy → Software and application security; Security and privacy → Software security engineering; Social and professional topics → Computing education; Social and professional topics → Computing education programs

Keywords and phrases Rust, Secure Coding Training, Secure Software Development, Serious Games, Gamification, Industry

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.16

Category Short Paper

Funding This work was partially supported by national funds through Fundação para a Ciência e Tecnologia, I.P. (FCT), ISTAR-Iscte Pluriannual Project UID/04466/2025 (`https://doi.org/10.54499/UID/04466/2025`).

Acknowledgements The authors would like to thank the industrial partners who supported this research.

1 Introduction

Cybersecurity incidents represent one of the most significant threats to modern organizations, frequently resulting in substantial financial losses and operational disruptions. In response, industry has established standards such as IEC 62443 that mandate secure development practices throughout the software lifecycle [7]. Despite advances in automated detection, developers remain responsible for writing secure code, and their security awareness directly impacts system resilience [13]. This human factor makes developer education a critical component of any security strategy. Moreover, recent research suggests that reliance on generative AI tools reduces developers' critical thinking effort [10], reinforcing the need for training that actively engages practitioners in reasoning about code security.

Serious games-interactive, game-based learning experiences-offer a promising approach to raising security awareness among developers [8], with challenge-based formats, such as Capture-the-Flag competitions, effectively engaging developers in industrial settings [8, 15].

Rust is a systems programming language whose ownership model, borrow checker, and type system eliminate entire classes of vulnerabilities common in C and C++ [9, 2]. However, developers can still introduce security issues through misuse of `unsafe` blocks, logic errors, improper input validation, or flawed concurrency patterns [16].

In this work, we focus on the human factor in secure Rust development by designing a serious game that trains industrial developers in Rust-specific secure coding. Our contributions are: (1) the design of secure coding challenges based on real-world Rust vulnerabilities mapped to CWEs, and (2) a structured evaluation protocol for validating the challenge design with industry security experts.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes our methodology. Section 4 presents the challenge design and planned evaluation protocol. Section 5 discusses design considerations, limitations, and future work. Section 6 concludes the paper.

2 Related Work

Prior work relevant to this paper spans three main areas: secure coding education and training needs, serious games and challenge design for technical learning, and Rust-specific software security.

Research on secure coding training shows that developers benefit from combining conceptual material with practical application [12]. Studies confirm that security knowledge directly affects software quality and that developer preparedness remains a concern [13, 17]. Furthermore, growing reliance on generative AI in development workflows has been shown to reduce cognitive effort in critical thinking tasks [10], highlighting the importance of training that requires active problem-solving. Large-scale surveys confirm that developers recognize gaps in their security knowledge [7], motivating the adoption of hands-on training approaches.

Serious games have been studied extensively as vehicles for effective learning, with systematic reviews identifying clear objectives, immediate feedback, difficulty progression, narrative context, and interactivity as important factors [11]. Prior work emphasizes aligning game design with professional tasks [4], supports game-based approaches for cybersecurity education [1], and validates structured, defense-oriented exercises in industrial settings [8]. Hybrid work environments have motivated flexible deployment of serious games [15], and using real-world vulnerabilities as a basis for exercises enhances engagement and perceived relevance [3].

Rust provides strong safety guarantees through its type system and ownership model, yet important security risks remain. Ecosystem-level analyses show that vulnerabilities in Rust crates are not uncommon, including logic errors, improper input validation, cryptographic misuse, and memory safety violations [16, 2]. These findings indicate that secure Rust development depends on developer understanding beyond what the compiler can enforce.

Despite the growing body of work on secure coding education, serious games, and Rust security, there is a lack of defense-based, interactive serious game challenges designed specifically for Rust secure coding training in industrial settings. The present work addresses this gap by transforming real-world Rust vulnerabilities into gamified challenges for professional developers [5, 6].

3 Methodology

Our methodology builds upon previous work on secure coding challenge design for industry [8, 4] and extends it to the Rust programming language. The overall approach consists of three main phases: (1) investigation of Rust security risks from literature and practice, (2) design of challenge scenarios and game structure, and (3) design of an evaluation questionnaire for industrial security experts.

In the first phase, we reviewed literature on security risks in the Rust ecosystem, particularly ecosystem-level analyses of vulnerabilities in Rust crates [16], complemented by the authors' practical experience with Rust in industrial contexts. For each vulnerability pattern, we assessed suitability for a challenge format based on: (a) relevance to typical Rust development tasks, (b) whether it can be reasonably simplified without losing educational value, and (c) amenability to a defense-oriented format where the developer must fix or prevent the issue.

In the second phase, based on the identified vulnerability patterns, we developed five challenge scenarios covering a range of vulnerability types and difficulty levels. For the storyline format, challenges are contextualized within a fictional company setting where the developer assumes the role of a security engineer receiving tickets describing codebase issues. Each ticket corresponds to a real vulnerability pattern, providing narrative motivation while maintaining technical authenticity.

In a third phase, to validate the approach, we plan to engage industry security experts to evaluate the proposed scenarios on criteria including technical accuracy, relevance, educational value, and engagement potential. The methodology follows an iterative approach: after initial evaluation, the design will be adjusted based on feedback, and subsequent reviews will assess the refined challenges. In this paper, we present our intended evaluation criteria, based on our experience in designing serious games, and teaching secure software development in an industrial context.

4 Results

In the current section we present the results obtained throughout the three phases, as described in the methodology section.

4.1 Rust Security Risks

From the vulnerability categories identified by Zheng et al. [16] and our industrial experience, we selected four CWEs that met all three selection criteria: CWE-190 (integer overflow), as arithmetic errors in Rust's `unsafe` contexts can bypass default overflow checks; CWE-125/CWE-787 (out-of-bounds read/write), representing memory safety violations common in sandbox and FFI code; CWE-20 (improper input validation), a pervasive issue even in safe Rust; and CWE-703 (improper handling of exceptional conditions), reflecting error handling patterns that lead to denial-of-service or undefined states. Table 1 presents the resulting five challenges.

4.2 Design of the Challenges

Based on the identified vulnerability patterns, we design interactive challenges with the following characteristics. Drawing on our experience and success designing serious games, we adopt a defense-oriented, code-entry format. The developer is presented with a vulnerable

Table 1 Mapping of challenges to CWEs, types, and difficulty levels.

ID	Challenge	Type	Difficulty	CWE(s)
C1	Integer Overflow	Individual	Easy	CWE-190
C2	Sandbox Breach	Individual	Medium	CWE-125 | CWE-787
C3	Input Validation	Storyline	Easy	CWE-20
C4	Improper Handling	Storyline	Easy	CWE-703
C5	Combined Vulnerabilities	Storyline	Medium	CWE-20 | CWE-703

Rust code snippet and must modify it to eliminate the vulnerability while preserving functionality. Challenges are delivered through Visual Studio Code (VSC), and submissions are evaluated by a backend pipeline combining unit tests and static analysis (e.g., `cargo clippy`) to verify correctness and security properties. Figure 1 illustrates the platform architecture.

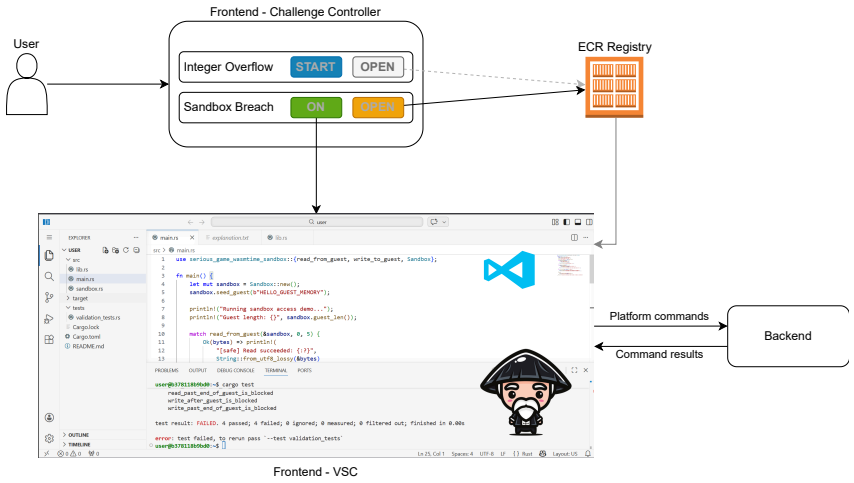

Figure 1 Architecture of the platform where challenges are deployed.

To illustrate, Listing 1 shows a simplified excerpt from challenge C2 (CWE-125/CWE-787), where the bounds check is insufficient-it verifies only the requested length without validating that the computed range stays within guest memory. Listing 2 shows the corrected version using safe bounds validation.

Listing 1 Vulnerable code: insufficient bounds check allows out-of-bounds read.

```
pub fn read_from_guest(
    sandbox: &Sandbox,
    offset32: u32,
    len: usize,
) -> Result<Vec<u8>, AccessError> {
    if len == 0 {
        return Err(AccessError::InvalidLength);
    }
    // Only checks length, not the full range
    if len > sandbox.guest_len() {
        return Err(AccessError::OutOfBounds);
    }
```

```
    let computed = sandbox.guest_base()
        .wrapping_add(offset32 as usize);
    unsafe {
        let ptr = sandbox.host_memory()
            .as_ptr().add(computed);
        let slice =
            std::slice::from_raw_parts(ptr, len);
        Ok(slice.to_vec())
    }
}
```

Listing 2 Corrected code: validates full range and avoids unsafe access.

```
pub fn read_from_guest(
    sandbox: &Sandbox,
    offset32: u32,
    len: usize,
) -> Result<Vec<u8>, AccessError> {
    if len == 0 {
        return Err(AccessError::InvalidLength);
    }
    let offset = offset32 as usize;
    let end = offset.checked_add(len)
        .ok_or(AccessError::OutOfBounds)?;
    if end > sandbox.guest_len() {
        return Err(AccessError::OutOfBounds);
    }
    Ok(sandbox.guest_slice()[offset..end].to_vec())
}
```

Since multiple valid fixes may exist for a given vulnerability, the backend evaluation relies on purpose-built unit tests and static analysis rather than exact output matching, allowing flexible acceptance of different correct solutions.

Based on previous work on scaffolding in secure coding challenges [8], we design an AI-powered hint system that dynamically generates personalized guidance at three progressive levels:

Level 1 – Conceptual: A general nudge towards the relevant secure coding principle, without referencing specific code.

Level 2 – Directional: Narrows focus to the relevant code area, suggesting what aspect to reconsider.

Level 3 – Explicit: A concrete indication of the fix, referencing the relevant Rust API or pattern, stopping short of the complete solution.

To illustrate, for challenge C2 (Sandbox Breach), the three levels might be:

L1: "Think about what happens when you trust a length value without considering where the access actually starts."

L2: "The bounds check might not be enough – consider whether the offset plays a role in the validity of the access."

L3: "You need to validate the full range (offset + length) against the buffer size, and watch out for arithmetic overflow when computing it."

Table 2 Expert evaluation dimensions.

Challenge Framing	Dimension	Statement for Evaluation
Individual	CVE Fidelity	The challenge adequately represents the associated CVE
	Standalone Clarity	The challenge can be understood without external context
	Real-world Representativeness	The challenge represents a real-world scenario
Storyline	Narrative Coherence	The storyline provides a coherent and believable context
	Inter-challenge Progression	The challenge fits logically within the storyline progression
	Contextual Added Value	The narrative context adds value to the learning experience
Both	Difficulty Appropriateness	The difficulty level is appropriate for the target audience
	CWE Coverage Adequacy	The challenges cover a sufficient range of vulnerability types
	Hint Quality	The hint system is helpful without revealing the solution
	Industry Relevance	The challenge reflects realistic industry scenarios
	Learning Potential	The challenge has potential to improve secure coding skills
	Engagement Potential	The challenge is engaging and motivating to solve

The implementation and validation of AI-generated hints constitutes a next step in this research.

A persistent "goal summary" command is always available, allowing the developer to revisit challenge objectives at any point. Upon submission, the developer receives immediate feedback on whether the solution passes security and functionality checks. After successful completion, a technical explanation is provided, including the canonical solution and why the original code was vulnerable. Unit tests are visible, ensuring transparency in evaluation criteria.

Challenges are organized into two difficulty levels: *easy* challenges target single, well-known vulnerabilities in isolation. In contrast, *medium* challenges introduce moderate complexity and may combine up to two vulnerability types in a single exercise. This progression reflects cognitive load management principles, presenting learners with incremental complexity. Similarly, the narrative framing of storyline challenges grounds tasks in authentic workplace scenarios (situated learning), while the scaffolded hint system reduces extraneous cognitive load by guiding attention progressively [14].

4.3 Expert Evaluation

To validate the challenge design, we define a structured evaluation protocol to be conducted with industry security experts from a large multinational technology company, targeting experts with experience in software security and familiarity with Rust development. This evaluation protocol is designed based on our experience in the design and evaluation of serious games for programming education in an industrial context.

Table 2 presents the designed evaluation framework, assessed on a 5-point Likert scale. This framework is used to evaluate the design of each challenge over different dimensions, grouped by three framings: *individual*, *storyline*, and *both*. Note that the CWE Coverage Adequacy dimension is assessed at the challenge set level rather than per challenge.

The dimensions capture both format-specific qualities (e.g., fidelity to source CVEs for individual challenges, narrative coherence for storyline challenges) and cross-cutting concerns (e.g., industry relevance, hint quality). The expected answers are on a 5-point Likert scale from 1 to 5, with 1 representing strong disagreement and 5 representing strong agreement with the statement.

5 Discussion

A fundamental premise of this work is that industrial developers have already mastered programming fundamentals, allowing the challenge design to focus exclusively on security and present sophisticated vulnerability patterns without requiring foundational instruction

on language mechanics. Previous work in academic settings [5] has shown that students often struggle with the language itself before engaging with security content; our work targets a population where this barrier does not exist.

The design process revealed several practical considerations. First, grounding challenges in real vulnerability patterns (via CWE mappings and adaptation to CVE) provides a concrete link between training and production concerns, which we expect to increase perceived relevance. Second, the distinction between storyline and individual formats emerged as a meaningful design axis: storyline challenges offer narrative context and progressive complexity, while individual challenges offer focused exercises with clear traceability to real incidents. Whether one format leads to better learning outcomes remains an open empirical question.

Relative to prior work on serious games for secure coding in C/C++ [4, 8], this work extends prior approaches in two ways: (1) challenge design tailored to Rust's ecosystem, targeting vulnerabilities that escape the compiler's static guarantees; and (2) the proposed use of an AI-powered hint system for personalized, context-sensitive guidance rather than static pre-authored hints.

Finally, we note that the challenge selection was informed by a simple study and the authors' experience which may introduce a certain bias. The current set of challenges covers only two difficulty levels, limiting the assessment of skills. Further, the evaluation protocol has not yet been executed; its results will inform iterative refinement of the challenge design.

In future work, we will conduct expert evaluation as described in Section 4.3 and refine the challenges. Subsequently, both storyline and individual formats will be deployed on the platform described in Figure 1, and we plan to compare their effectiveness through a developer survey measuring learning outcomes, engagement, and perceived relevance.

6 Conclusion

This paper presented the preliminary design of a serious game for secure Rust coding training in industrial settings. We mapped Rust vulnerability patterns to CWEs and developed challenge scenarios grounded in real-world issues. The design incorporates defense-oriented tasks, an AI-powered three-level hint system, automated toolchain-based evaluation, and progressive difficulty. Additionally, we defined a structured evaluation protocol for validation by industry security experts.

This work provides a foundation for expert evaluation, iterative refinement, and subsequent empirical comparison of both challenge formats, storyline-based and individual, with developer surveys constituting a later phase of this research.

References

1 Christopher J Berisford, Leighton Blackburn, Jennifer M Ollett, Thomas B Tonner, Chun San Hugh Yuen, Ryan Walton, and Olakunle Olayinka. Can gamification help to teach cybersecurity? In *2022 20th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–9, 2022. doi:10.1109/ITHET56107.2022.10031716.

2 Mohan Cui, Shuran Sun, Hui Xu, and Yangfan Zhou. Is unsafe an achilles' heel? a comprehensive study of safety requirements in unsafe rust programming. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3597503.3639136.

3 Denise Daniels, Joon Suk Lee, Hui Chen, and Kostadin Damevski. Utilizing real-world software vulnerabilities to enhance secure programming education. In *2024 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, 2024. doi:10.1109/FIE61694.2024.10893584.

4 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.

5 Tiago Espinha Gasiba, Andrei-Cristian Iosif, Santiago Suppan, Ulrike Lechner, and Maria Pinto-Albuquerque. Reflections on training next-gen industry workforce on secure software development. In *Proceedings of the 5th European Conference on Software Engineering Education*, ECSEE '23, pages 1–10, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3593663.3593665.

6 Tiago Espinha Gasiba and Ulrike Lechner. Raising secure coding awareness for software developers in the industry. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 141–143, 2019. doi:10.1109/REW.2019.00030.

7 Tiago Espinha Gasiba, Ulrike Lechner, Maria Pinto-Albuquerque, and Daniel Méndez. Is secure coding education in the industry needed? An investigation through a large scale survey. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 241–252, 2021. doi:10.1109/ICSE-SEET52601.2021.00034.

8 Tiago Gasiba, Ulrike Lechner, Maria Pinto-Albuquerque, and Alae Zouitni. Design of secure coding challenges for cybersecurity education in the industry. In Martin Shepperd, Fernando Brito e Abreu, Alberto Rodrigues da Silva, and Ricardo Pérez-Castillo, editors, *Quality of Information and Communications Technology*, pages 223–237, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-58793-2_18.

9 Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Safe systems programming in rust. *Communications of the ACM*, 64(4):144–152, 2021. doi:10.1145/3418295.

10 Hao-Ping (Hank) Lee, Advait Sarkar, Lev Tankelevitch, Ian Drosos, Sean Rintel, Richard Banks, and Nicholas Wilson. The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *CHI 2025*, April 2025. URL: https://dl.acm.org/doi/full/10.1145/3706598.3713778.

11 Werner Siegfried Ravyse, A. Seugnet Blignaut, Verona Leendertz, and Alex Woolner. Success factors for serious games to enhance learning: a systematic review. *Virtual Real.*, 21(1):31–58, March 2017. doi:10.1007/s10055-016-0298-4.

12 Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Training developers to code securely: Theory and practice. In *Proceedings of the 2024 ACM/IEEE 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability*, EnCyCriS/SVM '24, pages 37–44, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3643662.3643956.

13 Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Software security is your highest priority. Do your developers know that? *IEEE Security & Privacy*, PP:2–10, January 2026. doi:10.1109/MSEC.2026.3670641.

14 John Sweller, Jeroen J. G. van Merriënboer, and Fred Paas. Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review*, 31(2):261–292, 2019. doi:10.1007/s10648-019-09465-5.

15 Tiange Zhao, Tiago Gasiba, Ulrike Lechner, and Maria Pinto-Albuquerque. Thriving in the era of hybrid work: Raising cybersecurity awareness using serious games in industry trainings. *Journal of Systems and Software*, 210:111946, 2024. doi:10.1016/j.jss.2023.111946.

16 Xiaoye Zheng, Zhiyuan Wan, Yun Zhang, Rui Chang, and David Lo. A closer look at the security risks in the rust ecosystem. *ACM Trans. Softw. Eng. Methodol.*, 33(2), December 2023. doi:10.1145/3624738.

17 Shuqi Zuo, Qi Yang, Tianyi Gao, Zarin Tasnim Progga, and Jinjin Shen. How security coding knowledge impact software quality: An empirical study of stack overflow security issues. In *2025 25th International Conference on Software Quality, Reliability and Security (QRS)*, pages 727–738, 2025. doi:10.1109/QRS65678.2025.00076.