Cybersecurity

RESEARCH Open Access

Sifu - a cybersecurity awareness platform with challenge assessment and intelligent coach

Tiago Espinha Gasiba[1*], Ulrike Lechner[2] and Maria Pinto-Albuquerque[3]

Abstract

Software vulnerabilities, when actively exploited by malicious parties, can lead to catastrophic consequences. Proper handling of software vulnerabilities is essential in the industrial context, particularly when the software is deployed in critical infrastructures. Therefore, several industrial standards mandate secure coding guidelines and industrial software developers' training, as software quality is a significant contributor to secure software. CyberSecurity Challenges (CSC) form a method that combines serious game techniques with cybersecurity and secure coding guidelines to raise secure coding awareness of software developers in the industry. These cybersecurity awareness events have been used with success in industrial environments. However, until now, these coached events took place on-site. In the present work, we briefly introduce cybersecurity challenges and propose a novel platform that allows these events to take place online. The introduced cybersecurity awareness platform, which the authors call Sifu, performs automatic assessment of challenges in compliance to secure coding guidelines, and uses an artificial intelligence method to provide players with solution-guiding hints. Furthermore, due to its characteristics, the Sifu platform allows for remote (online) learning, in times of social distancing. The CyberSecurity Challenges events based on the Sifu platform were evaluated during four online real-life CSC events. We report on three surveys showing that the Sifu platform's CSC events are adequate to raise industry software developers awareness on secure coding.

Keywords: Cybersecurity, Awareness, Training, Artificial intelligence, Serious games, Secure coding, Static application security testing, Capture-the-flag, Software development in industry

Introduction

Over the last years, several attacks that target industrial control systems and cyberphysical systems have been identified. In 2010 Stuxnet, which attacks Programmable Logic Controllers, was uncovered; in 2014, the Havex malware, a Remote Access Trojan that contains code targeting industrial devices communicating over Open Platform Communications, was discovered. In the same year, BlackEnergy V3 attacked the Ukrainian power grid and energy distribution. More recently, in 2017, the Triton malware, which was coined *"the world's most murderous malware"*, was uncovered attacking the petrochemical industry in Saudi Arabia. The industry's financial impact due to these and other forms of malware has already exceeded 10 billion USD and affected more than 140 countries (Apextechservices 2017).

The Industrial Control System - Computer Emergency Response Team (ICS-CERT (Department of Homeland Security 2020a)) has been tasked with issuing ICS-specific alerts and advisories. Industrial Control Systems (ICS) alerts are information put out by the ICS-CERT with the intention *to provide timely notification to critical infrastructure owners and operators concerning threats or activity with the potential to impact critical infrastructure computing networks*. Figure 1) shows the number of ICS alerts and advisories issued per year by the ICS-CERT.

*Correspondence: tiago.gasiba@siemens.com
[1]Siemens AG Corporate Technology, Otto-Hahn-Rin 6, 81379 Munich, Bavaria, Germany
Full list of author information is available at the end of the article

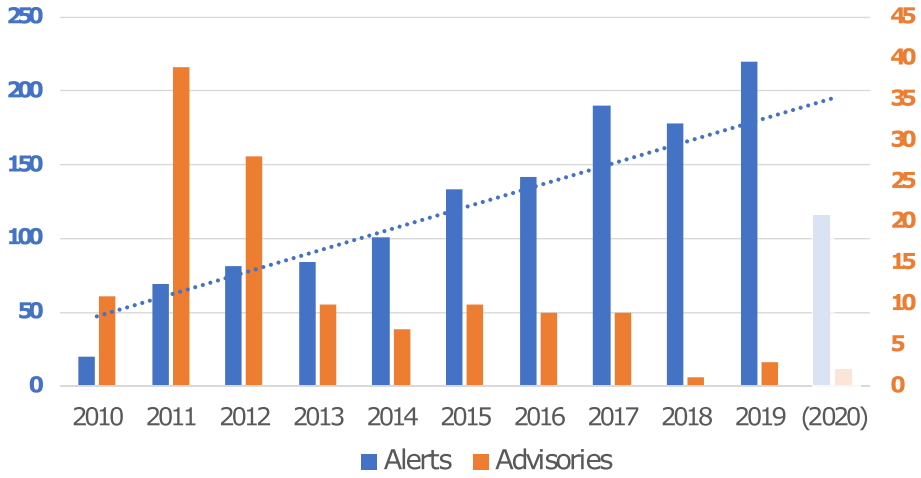

Fig. 1 ICS-CERT

Over the last decade, the number of security advisories issued per year has been steadily growing. Before 2014 less than 100 advisories per year have been issued, while from 2017 to 2019 more than 200 advisories per year have been issued. These numbers of security advisories correlate well with the observed increase in the number and sophistication of cyber-attacks to industrial control systems.

According to an estimation by the United States Department of Homeland Security (DHS), about 90% of the *reported security incidents result from exploits against defects in the design or code of software* (Department of Homeland Security 2020b). Related to this, a recent large-scale study by Patel el al. 2020 has shown that *more than 50% of software developers cannot spot vulnerabilities in source code* (Schneier 2020). These two factors considered together mean that: 1) special care must be exercised during software development, software developers, and 2) software developers lack awareness about secure coding.

Exploitation of low quality software can result in severe consequences for both customers, companies that produce the software (or product), and even unrelated third parties. The negative consequences are especially acute when the vulnerable software is deployed in critical infrastructures. In this scenario, the negative consequences can range from monetary losses to loss-of-life.

The work present aims to improve the current situation utilizing a serious game -that we coined Cyber-Security Challenges (CSC)- designed to raise awareness on secure coding, secure coding guidelines, and software development best practices of software developers in the industry. This work also presents the Sifu platform, a software tool developed to implement the CSC serious game.

Addressing code defects during software development

According to Mead et al. (2004), addressing security vulnerabilities in source code early in the software development life-cycle can save many costs. Their work presents an empirical model that shows the incurred costs of fixing software vulnerabilities at the following phases: requirements engineering, architecture, design, implementation, testing, deployment, and operations. The validity of this model is corroborated by Black (2004), which describes how addressing software defects early in the software development stages (in particular, by early involvement of the software testing team) can also lead to cost savings.

In an industrial setting, due to the requirements imposed by standards (e.g. ISO27k 2013, IEC 62443 2018, PCI/DSS 2015, BSI5.21 2014), the requirements engineering, architecture and design phases are typically well covered. The compliance to these standards is checked during industry audits. Recent data (Department of Homeland Security 2020b; Patel 2020), however, suggests that software defects (and vulnerabilities) are being introduced when the software is being developed - by software developers - in the implementation stage. In this early software development stage, we would like to address the software implementation stage in our work.

One possible method that can be used to reduce the number of introduced software vulnerabilities is Static Application Security Testing Tools (SAST) at the software implementation stage. However, these have been shown to *not perform well in detecting security vulnerabilities in the source code* (Goseva-Popstojanova and Perhinschi 2015), and consequently, additional mechanisms must be used. In our work, we concentrate on the human factor, the software developer, since the software developer ultimately writes the software by hand.

For the implementation, there is a vast number of possible programming languages. We have decided to focus our work on the C and C++ programming languages. Our motivation to choose this programming language is twofold: 1) because this programming language is being actively (and highly) used in the industry where the first author works as a consultant and 2) a recent study by WhiteSource (2019) shown that C and C++ are among the most vulnerable programming languages in terms of cybersecurity vulnerabilities.

Industrial standards and guidelines

In recognition of the importance of secure products and a consequence of the current move towards digitalization and higher connectivity, several large industrial players have joined together and committed to a document called the charter of trust (Siemens 2020). The Charter of Trust outlines ten fundamental principles that the partners vow to obey to address the issues inherent with cybersecurity. ICS relevant standards such as IEC 62443-4-1 2018 or ISO 27001 2013 mandate not only the implementation of secure software development life-cycle processes but also awareness training.

These standards (IEC 62.443 and ISO 27k) address security from a high-level perspective and are not specific enough about recommendations and policies to be followed in software development. Towards this goal, an industry-led effort was created, the Software Assurance Forum for Excellence in Code (SAFECode 2018), with the aim of *identifying and promoting best practices for developing and delivering more secure and reliable software, hardware, and services*.

In terms of the programming languages C and C++, due to its popularity in the industry, there exist several secure coding standards. Carnegie Mellon provides a popular secure coding standard - the SEI CERT Secure Coding Standard (Carnegie Mellon University 2019), which aims at *safety, reliability, and security of software systems*. Other popular (secure) coding standards include AUTOSAR 2017 for the automotive industry and the MISRA coding standard (Misra 2012; 2012) for embedded devices.

Another reason to focus on the software developer is that these standards contain undecidable rules, i.e., rules that cannot be automatically checked by an automaton (Kässtner et al. 2020). In this case, it requires human intervention to understand the software and decide the appropriate measure. This intervention is possible if the software developer is aware and knows the appropriate secure coding guidelines.

Serious games

A serious game (Dörner et al. 2016) is a game that is *designed with a primary goal and purpose other than pure entertainment*. Typically these games are developed to address a specific need such as learning or improving a given skill. Serious games are a well-established instrument in information security. They are discussed in de-facto standards as in the German Federal Office for Information Security - IT Baseline Protection (BSI Grundschutzkatalog) (Bundesamt für Sicherheit in der Informationstechnik 2019; Bundesamt für Sicherheit in der Informationstechnik 2020) as a mean to raise IT security awareness and increase the overall level of IT security.

A Capture-the-Flag (CTF) game is one possible instance of a serious game. CTF games were initially developed in the penetration testing community and are mostly used by pentesters, security professionals, academics, and hobbyists to improve their offensive skills. Votipka et al. 2018b argue in their work that CTF events can also be used as a means to improve security software development. In particular, they show that the participants to such events experience positive effects in improving their security mindset (i.e., defensive mindset). Davis et al., in 2014a, also discuss the benefits of CTF for software developers, and they argue that CTFs can be used to teach computer security and conclude that playing CTFs is a fun and engaging activity.

Playing cybersecurity (serious) games is gaining more and more attention in the research community (Rieb 2018; Rieb et al. 2017). In Frey et al. (2019), show both the potential impact of playing cybersecurity games on the participants and the importance of playing games as a means of cybersecurity awareness. They conclude that cybersecurity games can be useful to *build a common understanding of security issues*.

In their work, Simões et. al 2020 present several programming exercises for teaching software programming in academia. Their design includes nine exercises that can be presented to students to foster student motivation and engagement in academic classes and increase learning outcomes. Their approach uses gamification and tools to perform automatic assessment of submitted solutions to exercises. However, their work focus on the correct (functional) solution of the programming exercise and not on secure programming and security best practices aspects.

In a closer approach to a solution suitable to the industry, Gasiba et al., in 2019 perform requirements elicitation employing systematic literature review, interview of security experts, and elicit requirements from CTF participants and games performed in the industry. Their work focuses on identifying the requirements necessary for serious games events based on CTF to raise secure coding awareness of software developers in the industry. The newly derived type of event is called the CyberSecurity Challenges (CSC). Among other requirements, they conclude that CSC events

should focus on the defensive perspective instead of offensive.

In a further work (Gasiba et al. 2020b), the authors provide six concrete and different challenge types to be used in this kind of CSC event. One of these is the code-entry-challenge type. In this type of challenge, the player interacts through a web interface with a back-end by modifying vulnerable code until all the coding guidelines are fulfilled, thus solving the challenge.

Automatic challenge evaluation and intelligent coach

In Gasiba et al. (2020b), the concept of a code-entry-challenge is derived empirically, and no implementation hints are provided, only the core idea. The present work extends this previous work by providing a real-world implementation of a code-entry-challenge. In the following, we will present and discuss the CyberSecurity Challenges and introduce the Sifu Platform that is a code-entry-challenge for CSC events.

The goal of the Sifu Platform is to: 1) automatically analyze the solution submitted by the participant to the back-end, 2) determine if this solution contains vulnerabilities and fulfills the required functionality, 3) generate hints to the player if the solution does not achieve a pre-determined goal and finally 4) provide a flag (i.e., a unique code) which the player can use to gather points in the game. The correctness of the provided solution depends on the code following established, secure coding guidelines and secure programming best practices.

The generated hints are provided by an intelligent coach, which assists the player in solving the challenge. These hints are created using a simple artificial intelligence (AI) engine that provides automatic pre-programmed interactions with the player when the submitted solution fails to meet the secure coding criteria. These hints generated by the AI Engine (i.e., the intelligent coach) help the player solve the challenge playfully and help lower the frustration, increase the fun, and improve the gameplay's learning effect.

The core of the present work is to describe the intelligent coach platform and provide an evaluation of the Sifu Platform in terms of suitability to raise secure coding awareness. Three small surveys were developed and deployed with real players during four instances of the game to validate its suitability. The evaluation results show that the participants have fun using the platform and find it adequate to secure coding guidelines and secure software development best practices.

Previous work

The present work would not have been possible without the previous work of many colleagues and researchers. Figure 2 shows the seven main areas which have, in combination, influenced the current work: emerging needs, CTF, challenges, artificial intelligence, survey methodology, related theories, and IT security standards. The previous academic work on emerging needs gives motivational reasoning behind the current work. Non-academic work on emerging needs is related to an increasing company-internal demand and support by management in the development of novel awareness training methodologies. The rich literature on capture the flag (CTF), e.g. (Chung and Cohen 2014; Chung 2017; Bakan and Bakan 2018; Djaouti et al. 2011; Davis et al. 2014b; Cullinane et al. 2015; Hendrix et al. 2016; Sorace et al. 2018; Rieb et al. 2017; Rieb 2018; Votipka et al. 2018a), which is a serious game genre, discusses the recent scientific studies that have been performed on the usage of Capture-the-flag for IT security awareness training. Our work is also based on previous studies on challenges/exercises for teaching computer science, in particular related to IT security (Švábenskỳ et al. 2018; Hulin et al. 2017; Chapman et al. 2014; Mirkovic and Peterson 2014; Leune and Petrilli Jr 2017; Tabassum et al. 2018). The present work also makes use of artificial intelligence (AI) methods; in particular, it makes use of the lettering interview technique (Rietz and Maedche 2019). To evaluate our approach in terms of research questions, we follow best practices on survey design and follow standard existing analysis methodologies (Groves et al. 2009; Drever 1995; Harrell and Bradley 2009; Wagner et al. 2020). The main fundamental theories in which our work is based are on IT Security Awareness by Hänsch et al. and on Software Developer Happiness by Graziotin et al.

In their work, Graziotin et al. 2018 argue that *happy developers are better coders*. They show that developers that are happy at work tend to be more focused, adhering to software development processes, and following best practices. This improvement in software development concludes that happy developers can produce higher quality and more secure code than unhappy developers. Since they are experienced as fun events, the authors believe that CTF events can foster higher code quality and adherence to secure development principles.

Vasconcelos et. al 2020 have recently shown a method to evaluate programming challenges automatically. In their work, the authors use Haskell and the QuickCheck library to perform automated functional unit tests of students' challenges. Their goal is to evaluate if the students' solutions comply with the programming challenge in terms of desired functionality. One of the main limitations of this work is that the code to be tested should be free from side effects. The authors also focus on functional testing of single functions and do not address the topic of cybersecurity.

In Dobrovsky et al. (2016) and Brisson et al. (2012) describe an interactive reinforcement learning framework for serious games with complex environments, where a

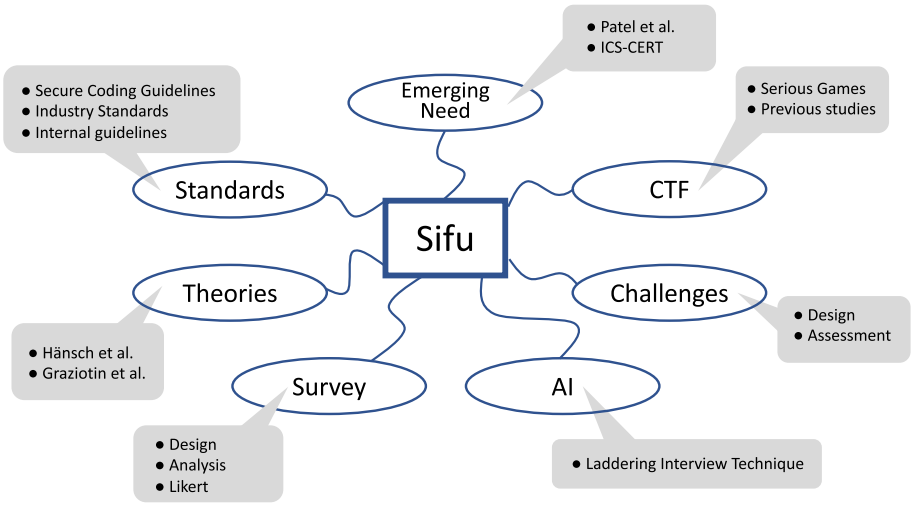

Fig. 2 Related Work

non-player character is modeled using human guidance. They argue that interactive reinforcement learning can be used to improve learning and the quality of learning. However, their work aims to train an algorithm to recreate human behavior employing machine learning techniques. In our work, we aim at training humans to write better and more secure code. Due to this fact, machine learning techniques are not applicable. Nonetheless, we draw inspiration from the conceptual framework, which we adapt to our scenario.

Rietz et al. 2019, show how to apply the laddering interview technique's principles to requirements elicitation. The laddering technique consists of issuing a series of questions based on previous system states (i.e., previous answers and previous questions). The questions generated are refined versions of previously issued questions as if the participant is climbing up a ladder containing more specific questions. Although this previous work applies in the field of requirements elicitation and does not focus on cybersecurity, the laddering technique principle can be adapted to a step-wise hint system, such as ours.

In the present work, we also use the concept of awareness or IT-security awareness as defined by Häensch et al. in 2014, in order to evaluate our artifact. In their work, they define awareness as having the following three dimensions: perception, protection, and behavior. The perception dimension is related to the knowledge of existing software vulnerabilities. The protection dimension is related to knowing the existing mechanisms (best practices) that avoid software vulnerabilities. Finally, the behavior dimension relates to the knowledge and intention to write secure code. We collect data from participants based on the three dimensions of awareness through a small survey. We use best practices in the design, collection, and processing of survey information given by Groves et al. 2009. Best practices from Crawley 2012 guide statistical analysis of the obtained results.

Contributions of this work

This work seeks to provide the following impact in the research community:

- introduces a novel method to automatically analyze player code submission in terms of secure coding guidelines and software development best practices,
- introduces an intelligent coach based on the laddering interview AI technique, and
- provides a preliminary analysis of the proposed architecture's suitability in terms of adequacy to raise secure coding awareness of software developers.

Although we intend to use the Sifu platform in a CSC event, this platform can also be used stand-alone, in remote and offline training scenarios. This offline scenario can be especially important if the players are spread over a large geographic area or have inherent restrictions on a face-to-face workshop, such as travel restrictions.

CyberSecurity challenges - a serious game for the industry

A CyberSecurity Challenge Event is a one-day event in which 10 to 30 software developers from industry participate. There are two types of events, suited for the software developers' different backgrounds: web-application and C/C++. In this work, we focus on challenges for the C/C++ programming language. These programming languages are widely used in the industry (IEEE Spectrum 2019), but are also among the most vulnerable in terms of cybersecurity vulnerability (WhiteSource 2019).

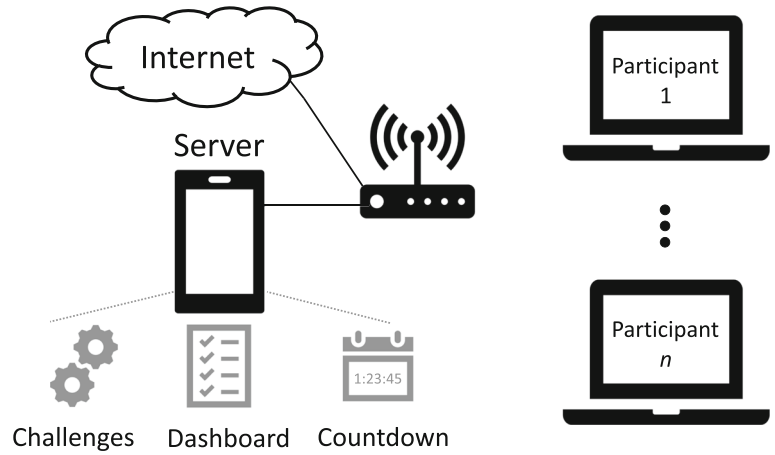

Fig. 3 Architecture of CyberSecurity Challenges infrastructure

During the workshop, the players form different teams that compete against eachother in solving the CSC challenges. These challenges address different topics related to secure coding and focus on secure coding guidelines (SEI-CERT (Carnegie Mellon University 2019), MISRA 2012 and AUTOSAR 2017).

Upon solving a challenge, the team is awarded a flag - i.e., a random-like code that is redeemed for points upon submitting to a dashboard. During the workshop, the players accumulate points by solving challenges. At the end of the event, the team with the most points wins the CSC game. However, the game intends that every participant profits from the game and that by concentrating on solving the challenges (Nakamura and Csikszentmihalyi 2014), the awareness of secure coding guidelines and secure coding best practices is exercised.

Figure 3 depicts the architecture, based on Gasiba et al. (2019), that we have conceptualized, designed, and deployed to implement the CSC game. It comprises a wireless access point that connects the players' computers, runs a local virtual machine, to a local server, and (optionally) connects to the internet. The server runs a dashboard (Chung 2020), a countdown website, and it also hosts the challenges. The players' local virtual machine can also host local challenges. The advantage of placing the different challenges in the participant's virtual machines is that they can be accessed after the game is finished.

Figure 4 shows the structure of a CyberSecurity Challenge, which consists of three phases: Phase 1 - *introduction*, Phase 2 - *challenge* and Phase 3 - *conclusion*. In Phase 1, an optional phase, the challenge, environment, and scenario are introduced. Furthermore, it is discussed the references to the secure coding guideline(s) about which the challenge is. In Phase 2, the player is presented with the challenge described in Phase 1, in the form of a project that needs to be solved by interacting with the Sifu platform. To solve the challenge, the player needs to adapt the code in the project in such a way as to be compliant with secure coding guidelines. While solving the challenge, the player is given several hints, depending on his progress in solving the challenge. These hints aid the player in solving

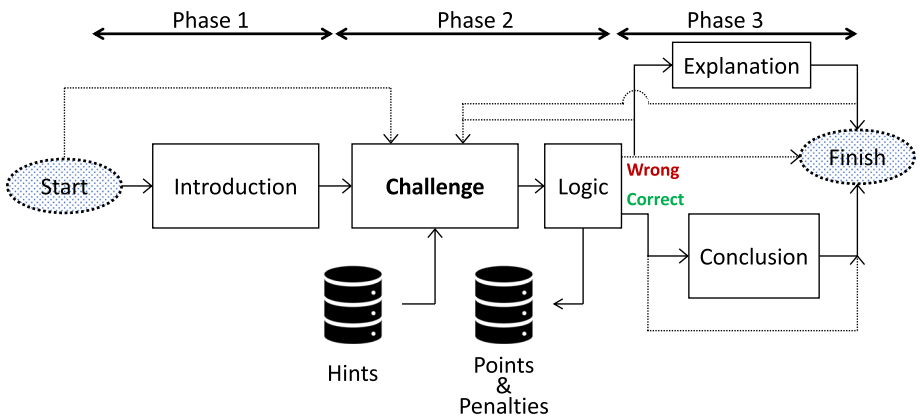

Fig. 4 Structure of a CyberSecurity Challenge

the challenge and serve to lower the frustration while playing the game. The players submit the proposed solution, for the challenge, to the back-end, where it is determined if the solution is acceptable or not. The player, optionally, might be awarded points or penalties at this stage, depending on the proposed solution. A detailed overview of this stage will be given in the following.

The processing in the last phase - Phase 3 - depends on the result of the previous phase. If the solution was wrong, the challenge is finished with an optional explanation of why the solution was not acceptable. If the solution was correct, the challenge is finished with an optional conclusion stage. This conclusion stage can include either additional questions (single or multiple-choice) or a debriefing. The debriefing contains a description of aspects related to the challenge, such as previous incidents, possible consequences of exploiting the vulnerability, the importance of the industry context's challenge, additional explanation of the secure coding guidelines related to the challenge. A player might or might not be able to have another attempt at solving the challenge, i.e., go to Phase 1 again, depending on the challenge configuration. If the player submits an acceptable solution, a flag is presented in Phase 3 (according to the CTF rules).

Sifu platform

In the following subsections, we present the research problem in terms of research questions, and our approach to solve them. In particular we describe the architecture of the proposed Code-Entry Challenge, which forms the Sifu Platform. We give details on how the Platform performs automatic assessment of solutions submitted by players and how an intelligent coach generates feedback messages, based on the results of the challenge assessment step. Furthermore, we provide a description of a real-world artifact and give a concrete example of a secure coding challenge. Finally, we also describe the surveys that we use to evaluate the approach.

Problem statement

In Gasiba et al. (2020b), the authors present a type of Challenge for CTFs in the industry called Code-Entry Challenge (CEC). The main idea, of this type of Challenge, is for the Player to be given a software development project, that contains code that does not follow secure coding guidelines (SCG), and secure software development best practices (BP), and contains security vulnerabilities. In this work, we target specifically ICS by using SCG and BP, which are specific to this field. The task of the Player is to fix the vulnerabilities and to follow SCG and BP. The Player should do this so that the original intended functionality is still fulfilled, in the new version of the code. The current work aims to solve these requirements using a platform that performs an automatic evaluation of the participant's code and guides the participant towards the final solution. The following research are then raised by considering these requirements:

RQ1: how to automatically assess the cybersecurity challenges, in terms of SCG and BP?

RQ2: how to aid the software developer, while solving the cybersecurity challenges?

RQ3: to which extent are cybersecurity challenges events, based on the Sifu platform, suitable as a means to raise secure coding awareness of software developers in the industry?

This work proposes to address RQ1, through a specialized architecture, to automatically assess the level of compliance to SCG and BP, by combining several state-of-the-art security testing frameworks, namely Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Runtime Application Security Protection (RASP). The functional correctness of the solution provided by the Player is evaluated using state-of-the-art Unit Testing (UT). We implemented this architecture, to automatically assess the cybersecurity challenges, through the Sifu platform; thus proposing an answer to RQ1.

To address RQ2, the authors propose to combine the output of the security testing tools, with an AI algorithm, to generate hints based on the laddering technique, thus implementing an intelligent virtual coach. The intelligent coach's task is to lower the participant's frustration during gameplay, and help the participant improve the code. This intelligent coach is embedded in our proposed Sifu platform. In this way, the Sifu platform with the intelligent coach contributes to answer RQ2.

Our proposed solution to address RQ1 and RQ2 is evaluated through two surveys: Survey 1 (S_1) and Survey 2 (S_2). Survey 1 (S_1) is composed of three quick questions asked to the participants, upon solving a challenge at the end of each game, but before obtaining the corresponding flag. Survey 2 (S_2) is composed of nine questions asked to the participants at end of the CyberSecurity Challenge event. To address RQ3, the authors have conducted an additional survey (Survey 3 - S_3) to evaluate the overall CyberSecurity Challenges event. Survey 3 (S_3) is composed of eleven questions asked to the participants at the end of the CSC event (in conjunction with S_2). The main difference, between S_2 and S_3, is that S_2 addresses specific questions related exclusively to the Sifu platform, while the questions in S_3 address the whole CyberSecurity Challenges event (including Challenges with Sifu platform).

The proposed solution, architecture, and design, herein described, contribute to answer research questions RQ1 and RQ2. The results of S_1 and S_2 contribute to evaluate

the Sifu Platform as a stand-alone platform for defensive challenges, i.e. contibute to RQ1 and RQ2. The results of S_3 contribute to evaluate the Sifu Platform as an integral part of CyberSecurity Challenges, in terms of the events' suitability to raise secure coding awareness of software developers in the industry, i.e. to address RQ3. Note that participation in S_1, S_2 and S_3 is voluntary, and therefore not all participants have decided to provide their answers.

Code-entry challenge platform architecture

Figure 5 shows the top-level view of the Sifu architecture.

In this figure, the "Player" represents the game participant (a human), and the "Project" represents a software project that contains vulnerabilities to be fixed by the Player. The "Analysis & Hints" (AH) component performs the core functionality:

- evaluates the submitted code (Project) in terms of SCG and BP
- indicates if the Challenge is solved or not and, if not solved
- generates hints to send back to the participant.

Previous interactions and generated hints are stored in the "State" component. During gameplay, the Player reads the Project and modifies it by interacting with a web editor interface. When the Player concludes the desired code changes, the Player submits it to the AH component (backend) for analysis.

A possible realization of the conceptual architecture is shown in Fig. 6. Interaction takes place between the Player and a web interface (web frontend), which connects to a web backend. The web backend is responsible for triggering the automated security assessment, collecting the AI engine's answer, and sending the answer back to the participant. Next, the Project submitted by the participant is saved into a temporary folder. Before the next step, the pre-processing of these Project files takes place. The goal of this pre-processing step is to inject the code necessary for unit tests. After adding auxiliary files (e.g., C/C++ include files) to the temporary project directory, the Project is compiled. If the compilation is successful, a functional test and security assessment is performed in a sandbox. All these results are then made available to an AI engine that determines if the Challenge is solved (i.e., if the solution is acceptable) and generates hints for player feedback otherwise. This feedback is collected by the web backend, stored in an internal database, and forwarded as the answer back to the participant's web browser.

Automatic security assessment

The security assessment performed on the Project is composed of the following steps: 1) Compilation, 2) Static Application Security Testing (SAST), 3) Unit Testing, 4) Dynamic Application Security Testing (DAST), and 5) Runtime Application Security Protection (RASP). In step 1, the Project is compiled; if there are compilation errors, these are reported to the AI component, and no further analysis takes place. Step 2 performs static code analysis. Note that in this step, the code does not need to be executed. Since the steps 3, 4, and 5 involve executing untrusted (and potentially dangerous) code, these are performed in a time-limited sandbox. The sandbox is very restrictive, e.g., it only contains the project executable and drops security-relevant capabilities (e.g., debugging and network connections are not allowed). Additionally, the executable is only allowed to run for a certain amount of time inside the sandbox. If this time is exceeded, the process will be automatically terminated. This avoids denial-of-service attacks by means of high CPU usage. Two types of Unit tests are executed: 1) functional testing - in order to guarantee that the provided code is working as intended (e.g., in the challenge description), and 2) security testing - in order to guarantee that typical vulnerabilities are not present in the code (e.g., buffer overflow). Security testing is done using self-developed tests and also using state-of-the-art fuzzing tools. Steps 4 and 5 perform several dynamic security tests. Table 1 lists the tools used in each of these components (in italic). The same table also lists additional potential tools that the authors are considering integrating into the Sifu Platform in a future work, and are given here for reader reference and completeness.

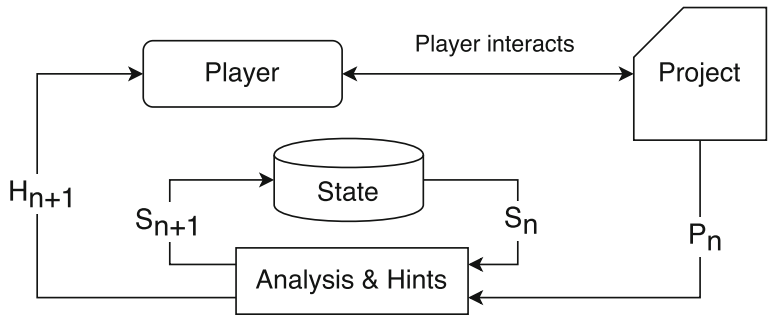

Fig. 5 Conceptual game overview: interaction and components

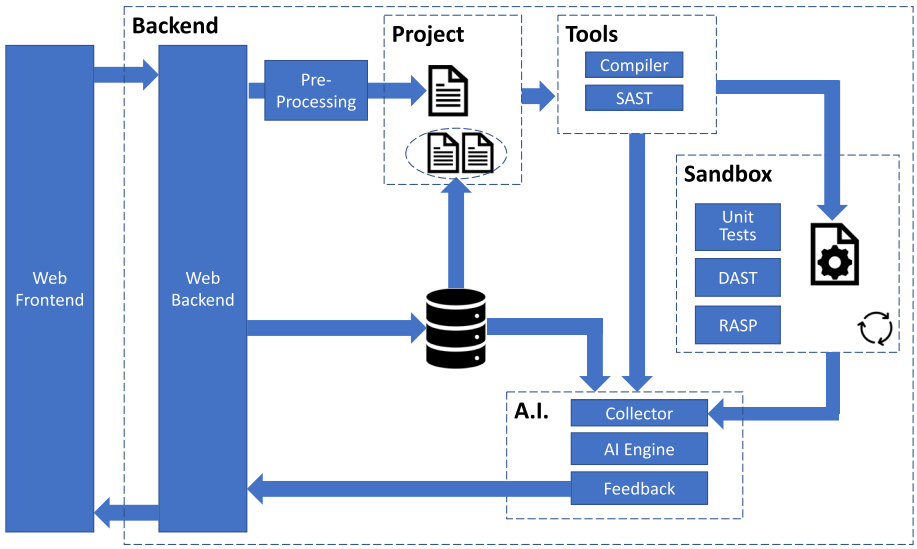

Fig. 6 Detailed architecture: the Sifu Platform

In this table, the open-source components used in the Sifu platform are marked with "OS".

Intelligent coach with AI technique

The AI component, shown in Fig. 6, collects the results of the previous analysis steps, runs an AI engine based on the laddering technique, and generates the feedback to be sent back to the participant. Figure 7 shows the implementation of the AI engine using the laddering technique (Rietz and Maedche 2019).

As previously detailed, the automated assessment tools perform several tests to determine the existing software vulnerabilities present in the Project. These are collected in textual form (e.g., JSON and XML), and normalized before being processed by the AI engine. The two most essential tests, resulting from the security assessment, are related to compilation errors (e.g., syntax errors), and functional unit testing. The participant's solution will be rejected if the code does not compile, or is not working (functioning) as intended. When both these tests pass, the artificial engine uses the security tests, SAST, DAST, and RASP tools to generate hints to send to the participant.

A combination of findings from these tools forms a vulnerability. These findings and vulnerabilities are mapped to SCG and BP. In Fig. 7, each horizontal path (ith row) corresponds to a ladder, and a specific combination of vulnerabilities or static events found in the source code. Each path is also assigned a priority $p(i)$, based on the criticality of the SCG and vulnerabilities. These priorities are assigned according to the ranking of secure coding guidelines, as presented in Gasiba et al. (see Gasiba et al. (2020a)). Higher-ranked secure coding guidelines are given higher priorities, and lower-ranked secure coding guidelines are given lower priorities. The AI engine then selects a path (corresponding to one ladder) based on the highest rank finding.

Table 1 Security assessment tools

Component	Tools
Compiler	GCC v10.1 (Stallman 2002) (OS), Clang 9.0.0 (OS) (Lattner 2018)
	AbsInt RuleChecker (AbsInt 2020a), SonarQube (SonarSource 2020), Pc Lint (Gimpel 2020)
	cppcheck (OS) (Marjamäki 2017), fbinfer (OS) (FaceBook 2020), semgrep (OS) (R2C 2020)
SAST	Clang-Tidy (OS) (The Clang Team 2020), *FlawFinder (OS)* (Wheeler 2013), *Frama-C (OS)* (Baudin et al. 2020)
	Graudit (OS) (Wireghoul 2020), *CMetrics (OS)* (MetricsGrimoire 2020), *ESBMC (OS)* (ESBMC 2020; Gadelha et al. 2018)
	TScanCode (OS) (Tecent 2020), *Ikos (OS)* (NASA-SW-VnV 2020)
DAST	AbsInt Astrée (AbsInt 2020b), Valgrind (OS) (Valgrind Developers 2010), Helgrind (OS) (Valgrind Developers 2020)
RASP	Address Sanitizer (OS) (Google 2020b), Leak Sanitizer (OS) (Google 2020c),
	Thread Sanitizer (OS) (Google 2020d)
Unit Test	ATF (OS) (JMMV 2020a), Kyua (OS) (JMMV 2020b), AFL (OS) (Google 2020a)

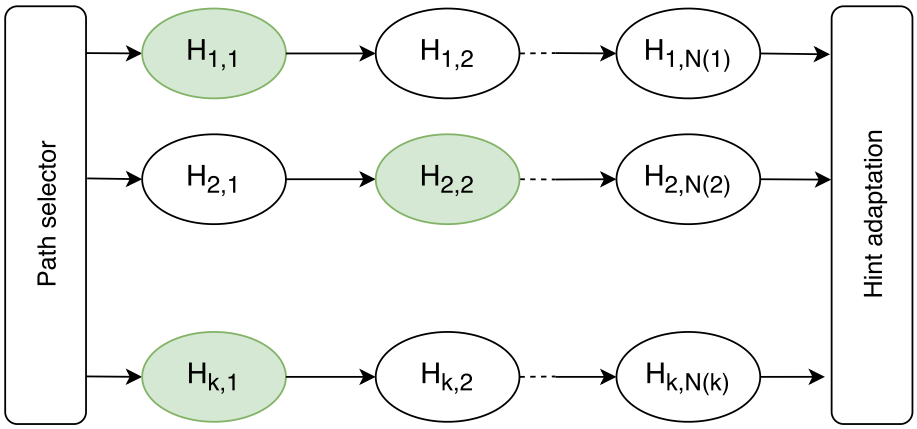

Fig. 7 Laddering technique to generate hints

The chosen hint H_{n+1} depends on the ladder and on the previous hint level sent to the participant on the ladder, as given by the system state. If there are no more hints in the ladder, no additional hint is sent to the Player.

Table 2 shows an example of hints, provided by the intelligent coach's AI engine, corresponding to an "undefined behavior" path. The lower level hints are generic and give background information for the participant. The highest level hint contains exact information on how to solve the problem, thus revealing the solution.

Finally, the Feedback component (part of the AI component in Fig. 6) formats and enriches the AI Engine's selected hint with project-specific information, and sends it to the Web Back-End component to be presented to the Player. To foster critical thinking, the authors have also implemented a hint back-off. This back-off system implements the following rule: 1) no hint is provided to the Player during 4 minutes after the backend has sent a hint to the Player, and 2) no hint is given until the number of code submissions, since the previous hint sent by the backend to the Player, is equal to 3 submissions (i.e., no hint will be given to the Player who is brute-forcing the hint system).

Note that the feedback component, not only fosters critical thinking by the Player, but can also be used to train the Player with the usage of static code analysis tools. However, further investigation of this aspect is needed in the future.

Table 2 Example of hint ladder with six levels

Level	Hint text
1	Maybe you could have a look at the following link: < *link* >
2	The following link < *link* > contains additional information
3	You can either use a secure function or locally disable compiler optimization
4	Have a look at Annex K of the C standard here: < *link* >
5	You can also consider turning on/off optimization: < *link* >
6	Note that memset_s is optional in the standard...
7	We provide you with memset_s if you include memset_s.h in your code.
8	Have a look at a possible solution to the challenge: < *link* >

Real-World artifact

Figure 8 shows the web interface of a real-world implementation of the Sifu platform. The machine where the Sifu platform was deployed was an AWS instance of type T3.Medium (2 CPUs with 4Gb RAM and network connection up to 5Gb/s). In order to install the required tools, a hard-disk of 40Gb was selected. The Sifu platform itself is developed in Python 3.8 using Flask.

On the left, the Player can browse the Project and select a file to edit; the file editor is in the center, and on the right are the hints that the Player receives from the backend. The upper part contains buttons which include the following functionalities: *Submit* - to submit the Project for analysis, *Reload* - to reload the Project from scratch, and *Report Challenge* - to report problems with the Challenge to the developers. Note that, when a player finishes a challenge successfully, it is taken to an additional page with discussions on the impact of the vulnerability and additional closing questions (e.g., on which secure coding guidelines have not been taken into consideration).

Example of a secure coding challenge

Figure 9 (left) shows the first phase of a Sifu Challenge related to CWE-14 (MITRE 2020a). This vulnerability and the corresponding secure coding guideline (MSC06-C) is about dead-store removal. Table 3 contains information about CWE-14, as well as the other six Common Weakness Enumerations used in Sifu Challenges.

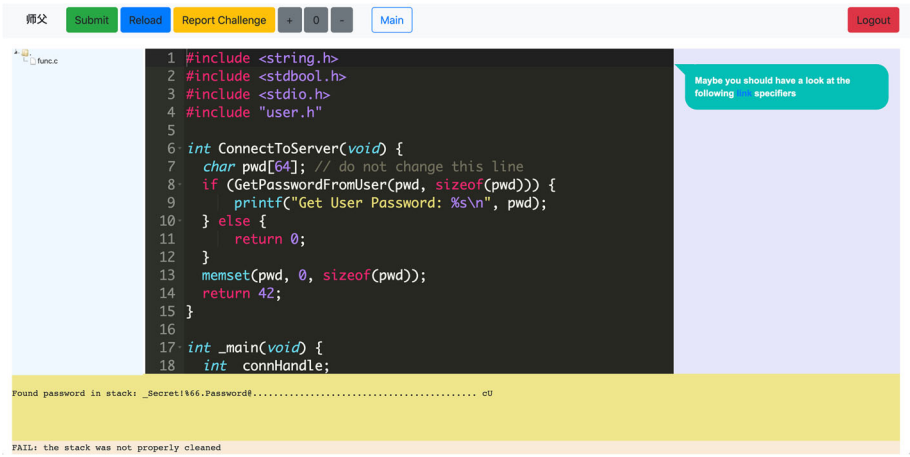

Fig. 8 Sifu Web Interface

When C-code is compiled with optimization turned on, a compiler can eliminate parts of the original code during compilation. In particular, a compiler can eliminate memory clearing functions (memset) of stack variables, if a function does not use the memory locations anymore until returning from the function. When the function returns, the stack's allocated memory must not be accessed anymore by any other function; otherwise, this would result in undefined behavior. As such, assuming that the memory cannot be used, the compiler is free to remove any memory clearing functions, since this cannot have any more side-effects, according to the C-Standard.

In the introduction to the CWE-14 Challenge, a short background information is given to the Player. Also, the Player's task is clearly explained: to re-write the code, following secure coding guidelines, and to make sure that the sensitive memory locations are cleared, before returning from the C-function.

The second phase of the Challenge consists of the Player interacting with the Sifu platform. Figure 10 shows the C-code that is presented to the user. This code contains the vulnerable function, as discussed in the introduction to the Challenge. To solve this Challenge, the Player needs to either: 1) replace the call to memset with a call to memset_s, or 2) disable compiler optimization for the ConnectToServer function with a compiler #pragma. In order to assist the Player with this task, the Sifu platform provides hints to the Player, which aid towards the correct solution (see Table 2). Upon solving the Challenge, the Player is given information about the vulnerability and possible consequences thereof. Figure 9 (right) shows the information provided to the player upon completion of the challenge. The Player is also given links to further company-internal or company-external references, related to the Challenge.

Evaluation of real-world artifact

The Sifu platform, containing seven different challenges, as shown in Table 3, was evaluated during four different CSC events. Table 4 shows a summary of all these events,

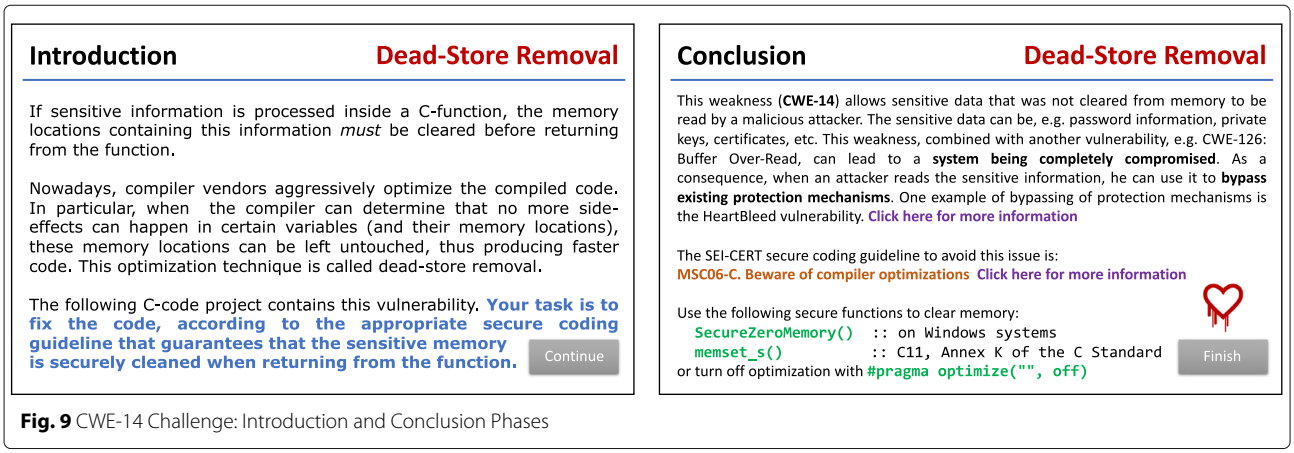

Fig. 9 CWE-14 Challenge: Introduction and Conclusion Phases

Table 3 Common weakness enumeration used in Sifu challenges

CWE	Ref.	Related SCG	Description
CWE-14	(MITRE 2020a)	MSC06-C	Compiler Removal of Code to Clear Buffers
CWE-77	(MITRE 2020b)	ENV33-C	Improper Neutralization of Special Elements used in a Command
CWE-121	(MITRE 2020c)	ARR38-C STR31-C	Stack-based Buffer Overflow
CWE-242	(MITRE 2020d)	POS33-C	Use of Inherently Dangerous Function
CWE-338	(MITRE 2020e)	MSC30-C	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
CWE-676	(MITRE 2020f)	CON33-C ENV33-C	
		ERR07-C ERR34-C	Use of Potentially Dangerous Function
		FIO01-C MSC30-C	
		STR31-C	
CWE-758	(MITRE 2020g)	ARR32-C ERR34-C	
		EXP30-C EXP33-C	
		FIO46-C INT34-C	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior
		INT36-C MEM30-C	
		MSC14-C MSC15-C	
		MSC37-C	

CWE: Common Weakness Enumeration, **SCG.**: SEI-CERT Secure Coding Guideline

which took place in an online format in June 2020 and July 2020. The ages of the participants' ranged between 20 and 50 years old. In the first event, from the 15 participants from Germany, 8 were from academia (7 computer science students and one assistant professor), and 7 were software developers from the industry. In the remaining events, all participants were software developers from the industry.

During the first event, the participants were allowed to experiment with the platform for as long as they liked. The total time it took the participants to experiment was from 15 min to 45 min. The last three events were embedded in a CSC event, which lasted one entire working day (8 h). These last CSC events consisted of a) a one-hour introduction to the event and explanation of the game

```
#include <string.h>
#include <stdbool.h>
#include <stdio.h>
#include "user.h"

int ConnectToServer(void) {
    char pwd[64]; // do not change this line
    if (GetPasswordFromUser(pwd, sizeof(pwd))) {
        printf("Get User Password: %s\n", pwd);
    } else {
        return 0;
    }
    memset(pwd, 0, sizeof(pwd));
    return 42;
}

int _main(void) {
    int connHandle;

    if (connHandle=ConnectToServer()) {
        // handle connection to the server
    } else {
        // handle error
    }
    return 0; // _main: do not change this line
}
```

Fig. 10 Project Code for CWE-14 Challenge

Table 4 Overview of CSC Events with Sifu Platform

No.	Date	Participants	NP	Where	Dur.	C	A	I	Survey
1	16 Jun 2020	15: Germany	15	Online	1h	2	8	7	S_1, S_2
2	8 Jul 2020	2: China, 15: Germany, 3: India, 1: UK	21	Online	8h	2	0	21	S_1, S_2, S_3
3	22 Jul 2020	20: Germany	20	Online	8h	2	0	20	S_1, S_2, S_3
4	31 Jul 2020	2: China, 8: Germany, 3: India, 2: UK	15	Online	8h	2	0	15	S_1, S_2, S_3

NP: nr. of players, **Dur.**: Duration, **C.**: No. Coaches, **A.**: No. Academia, **I**: No. Industry

mechanics, b) the main CSC event where the competition took place, c) announcement of the winner and collection of survey feedback, and finally d) walk-through of selected challenges together with all the participants.

The first part of the CSC event was to ensure that all the participants have access to the required virtual machines, that the individual teams are formed, that the participants are informed about how the game is played, and that they know how to use the Sifu platform. Since the events extended over an entire day, the individual teams had to decide on their lunch break strategy. Some teams decided to have a split-strategy (lunch-break is split into two, where in the first part, half of the team members take time off while other continues playing and vice-versa), while other teams decided to have a complete break (all team members stopped playing during lunch-break) and, finally one team decided to take no lunch-break. After the main event, a small ceremony announcing the winning team takes place with a small feedback round. In the feedback round, the coaches interact with the players to determine which challenges were more complicated and need to be further discussed. In the last part of the CSC event, the coaches performed a walk-through of selected challenges. This walkthrough is based on the collected feedback from the participants in the previous step.

During the gameplay, when successfully solving a challenge, the participants were asked (through the Sifu web interface) to rate the Challenge based on three questions, which we call Survey 1: S_1. During the first event, upon completing the experiment, the participants were asked to fill out another survey, which we call Survey 2: S_2. Finally, for the last three events, during the feedback phase, the participants were asked to fill out a survey that was an extended version of S_2. At the end of the CSC event, the participants were also asked to rate the overall event with Survey 3 - S_3. The questions asked to the participants are shown in Table 5.

The goal of S_1 is to get immediate and short 3-question feedback on the challenges contained in the Sifu platform. In particular, the participants were asked to rate the Challenge, and rate how well they can recognize and fix the vulnerability in production code. Survey S_2 contains a question that evaluates the Sifu platform itself. This survey, together with S_1, is used to evaluate the suitability of the Sifu platform, as a means to automatically assess the cybersecurity challenges (RQ1); as well as a means to help software developers, while solving cybersecurity challenges (RQ2).

We use the definition of IT Security Awareness from Hänsch et al. 2014, with its three dimensions (Behaviour, Protection, and Perception) as the theoretical framework to develop the questions. Finally, the survey S_3 was developed to evaluate the Sifu platform, and the entire CSC event. The survey questions in S_3 are based on the industry's teaching experience, in secure coding, by the first author and on the experience gathered from previous CSC events. This survey's primary goal is to determine if the participants agree that the CSC event helps them be more prepared to deal with secure coding issues at work.

Although all participants were kindly asked to answer the surveys S_1, S_2, and S_3, not everyone has decided to participate, since taking part in this study was not mandatory. Answers to the survey questions were based on a 5-point Likert (Joshi et al. 2015) scale: SD - Strongly Disagree, D - Disagree, N - Neutral, A - Agree, and SA - Strongly Agree.

Results

This section presents the individual results of the different surveys S_1, S_2, and S_3. Additionally, we perform a closer analysis of the results of S_2 and S_3 and finally relate the overall results to the research questions. We conclude this section with a brief discussion on the threats to validity. All the data presented was processed using RStudio version 1.2.5019 (R Core Team 2019).

Challenge feedback - survey S_1

Figure 11 shows the aggregated results of the challenge rating questions of survey S_1. These results were collected through 44 solved challenges during the four CSC events, as shown in Table 4. These four events counted with the participation of 71 players with an average team size of 4 players per team. Note that the data collected only

Table 5 Questions for surveys S_1, S_2 and S_3

Survey	QID	T	TC	Question
S_1	Q1	-	-	Please give an overall rating to the challenge
	Q2	-	-	How well could you recognize the vulnerability in the code?
	Q3	-	-	How well can you fix this problem in production code?
S_2	F1	(Graziotin et al. 2018)	HP	My overall experience with the platform was positive
	F2	(Hänsch and Benenson 2014)	BE	The Sifu platform helps me to improve my secure coding skills
	F3	(Hänsch and Benenson 2014)	PE	Solving challenges in the Sifu platform helps me in recognizing vulnerable code
	F4	(Hänsch and Benenson 2014)	PR	Solving challenges in the Sifu platform helps me in understanding consequences of exploiting vulnerable code
	F5	(Graziotin et al. 2018)	HP	Solving challenges in the Sifu platform makes me overall happy
	F6	(Hänsch and Benenson 2014)	BE	Challenges in the Sifu platform help me to practice secure coding guidelines
	F7	(Graziotin et al. 2018)	HP	I find the Sifu platform adequate as a means to raise awareness on secure coding
	F8	(Graziotin et al. 2018)	HP	The examples in the Sifu platform are clearly presented
	F9	(Graziotin et al. 2018)	HP	It is fun to solve challenges in the Sifu platform
S_3	X1	(Hänsch and Benenson 2014)	PE	I have learned new techniques and principles related to secure software development
	X2	(Hänsch and Benenson 2014)	PR	The challenges help me to understand possible consequences of security breaches
	X3	(Hänsch and Benenson 2014)	BE	I feel I am better prepared to handle secure coding-related issues at work
	X4	(Hänsch and Benenson 2014)	BE	The challenges help me to understand the need to have a well defined secure software development lifecycle activities
	X5	(Hänsch and Benenson 2014)	PE	I've learned about new issues related to security that can occur during software development through playing the challenges
	X6	(Graziotin et al. 2018)	HP	The help from the coaches was adequate
	X7	-	WK	I feel more prepared to understand the output of static application security testing tools
	X8	-	WK	Through playing the challenges, I know where I can find information about secure coding guidelines
	X9	(Hänsch and Benenson 2014)	PR	Through playing the challenges I understand the importance of using secure coding guidelines
	X10	(Hänsch and Benenson 2014)	BE	Through playing the challenges I feel my practical secure coding skills have improved
	X11	(Graziotin et al. 2018)	HP	The challenges are related with my daily work in my company

T: Theory, **TC**: Theory Construct, **PR**: Protection, **PE**: Perception, **BE**: Behavior, **HP**: Happiness, **WK**: Work

contains answers directly given by the teams as not everyone has decided to take part in the current study.

The average values and standard deviation are the following: Q3 3.86 ($\sigma = 1.11$), Q1 3.82 ($\sigma = 1.13$), Q2 3.80 ($\sigma = 1.19$). In general, the participants have agreed with all the asked questions, namely that the challenges are good (Q1), that they were able to recognize the vulnerability in the code (Q2) and that the participants can fix this problem in production code (Q3). Note that all the questions show a similar average agreement rating and standard deviation.

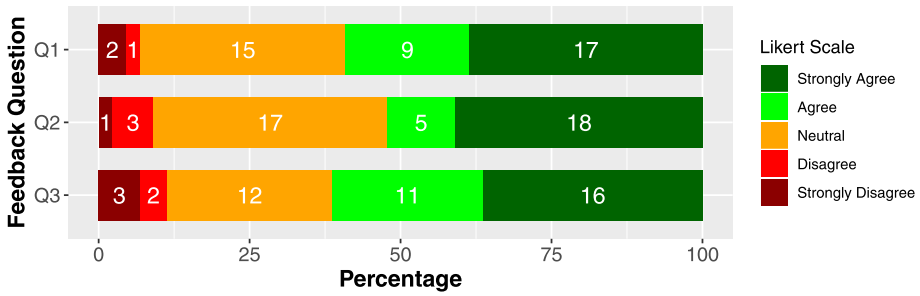

Fig. 11 Evaluation of Challenges in Sifu Platform - Survey S_1 - Overview

A break-down of these numbers towards the different challenges (see Table 3) is shown in Figs. 12, 13, and 14, for Q1, Q2 and Q3 respectively.

Except for the challenge CWE-676 (Use of potentially dangerous function), all the challenges clearly show positive feedback in Q1 - overall challenge rating (see Fig. 12). We note that, although the average agreement results for challenges CWE-14 (Compiler removal of code to clear buffers) and CWE-758 (Reliance of undefined, unspecified, or implementation-defined behavior) have a positive rating, they also have a low number of answers, in comparison to the other CWEs.

Figure 13 shows the results for Survey S_1, question Q2. Except for CWE-676, all collected results show a positive agreement. Also, for CWE-14 and CWE-758, although only four answers are considered, these show an average agreement, which is positive.

Finally, Fig. 14 shows the results for Survey S_1, question Q3. Again, for CWE-676, we observe a low rating and a low number of answers. Except for this challenge, the next challenge with lower agreement on Q3 is CWE-77, but still with a positive rating.

Table 6 shows a summary of the weighted average agreement for Survey S_1, Q1, Q2, and Q3. In general, we can observe that the weighted average agreement for Q2 and Q3 is correlated with the challenge rating in Q1. The higher the rating in Q1, the higher the values in Q2 and Q3 and vice-versa. Therefore, question Q1 can give a good indication, for practitioners, on which challenges need to be improved (e.g., better introduction and better hints).

Survey for Sifu platform - survey S_2

Figure 15 shows the results of the survey S_2 with a total of 25 answers from 71 participants, i.e., the participation rate was 35.2%. We observe that most of the collected results (i.e., more than 50% of the answers) for all the questions (F1 to F9) are either Agree or Strongly Agree. These results give a good indication that the Sifu platform aids developers to solve the cybersecurity challenges, helping to improve their awareness of secure coding, and presents a positive experience overall. The average weighted agreement values and standard deviation, sorted from the highest to the lowest ranking, are the following: F6 4.24 ($\sigma = 0.44$), F9 4.24 ($\sigma = 0.93$), F8 4.12 ($\sigma = 0.88$), F1 4.08 ($\sigma = 0.76$), F5 4.08 ($\sigma = 1.19$), F2 4.04 ($\sigma = 0.35$), F3 3.84 ($\sigma = 0.55$), F7 3.84 ($\sigma = 0.80$), and F4 3.68 ($\sigma = 0.80$).

We observe that 6 out of 9 questions have an average weighted agreement of more than 4 Likert points. The results also show that the highest-ranked question is F6 - the Sifu platform helps to practice secure coding

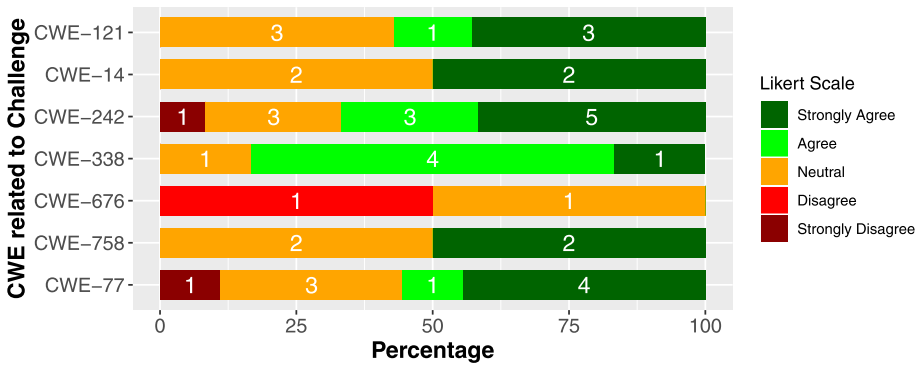

Fig. 12 Evaluation of Challenges in Sifu Platform - Survey S_1, Q1

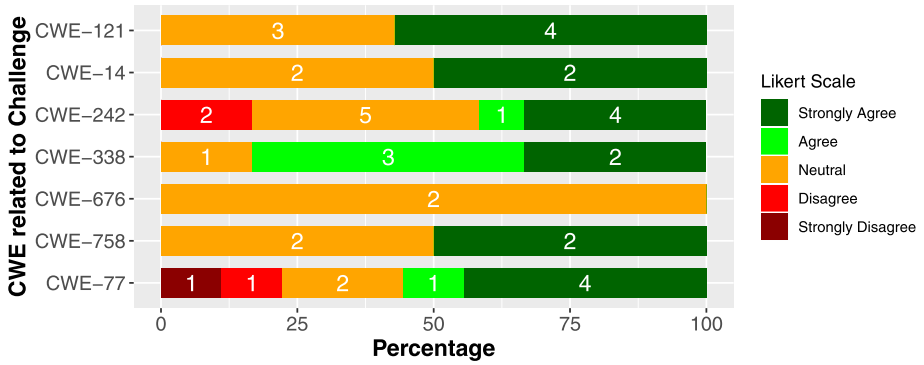

Fig. 13 Evaluation of Challenges in Sifu Platform - Survey S_1, Q2

guidelines. The next highest-ranked question is F9, which indicates that the participants find that solving the Sifu platform's challenges is a fun activity. The next highest-ranked question is F8, which is related to the usability of the platform. From this, we can conclude that the Sifu platform's challenges are well presented and intuitive for the participants. However, further research on the usability of the platform is required.

Although still ranked positive, the lowest positive result was for F4 - Sifu platform helps understand the consequences of exploiting vulnerable code. This result is expected as the challenges are presented from the defensive perspective, and the players never get to see an exploit in action. However, the authors think that this value is still high due to the Phase 3 of the CSC challenges - the conclusion - where the consequences of exploits, and previous real-world cases, are presented and discussed.

Survey for CSC event with Sifu platform - survey S_3

Figure 16 shows the results of the last survey, administered to the participants of the CSC events two, three, and four, as shown in Table 4. A total of 10 survey answers were collected, out of the 71 participants, i.e., the participation rate for Survey S_3 was 14.1%. The lower number of participants concerning S_1 and S_2 has to do with survey S_3 being completed, by the participants, after the end of the CSC event, and the fact that participation in the survey is not mandatory.

The average weighted agreement values and standard deviation, sorted from the highest to the lowest ranking, are the following: X6 4.90 ($\sigma = 0.32$), X9 4.60 ($\sigma = 0.52$), X4 4.40 ($\sigma = 0.70$), X8 4.40 ($\sigma = 0.70$), X1 4.30 ($\sigma = 0.67$), X2 4.30 ($\sigma = 0.67$), X5 4.30 ($\sigma = 0.48$), X10 4.10 ($\sigma = 0.57$), X3 4.00 ($\sigma = 0.67$), X7 3.90 ($\sigma = 0.74$)and X11 3.70 ($\sigma = 0.95$). Nine, out of the eleven questions, have obtained an average weighted agreement score of 4 or more Likert points. The highest-ranked question is X6 - help from the coaches was adequate. This result shows the importance that real (human) coaches have in making the CSC event a successful event. This contribution to success includes the coaching provided, during the introduction and conclusion phases. The next highest-ranked question is X9 - understanding the importance of secure coding guidelines. Since the entire event is directed towards exercising awareness on secure coding guidelines, it is also not surprising that this question is ranked with a high positive value. Finally, in the top three is X4 - understand secure software development lifecycle activities. This question's

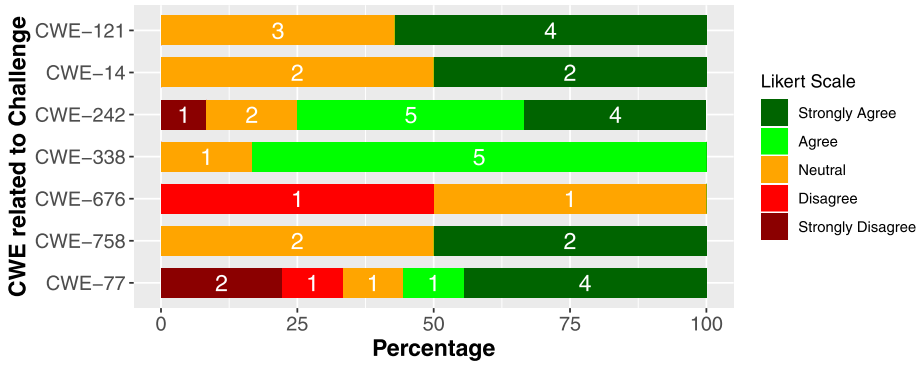

Fig. 14 Evaluation of Challenges in Sifu Platform - Survey S_1, Q3

Table 6 Sifu Challenge vs Survey S_1 - Average Agreement

CWE	Q1	Q2	Q3
CWE-14	4.00	4.00	4.00
CWE-77	3.78	3.67	3.44
CWE-121	4.00	4.14	4.14
CWE-242	3.92	3.58	3.92
CWE-338	4.00	4.17	3.83
CWE-676	2.50	3.00	2.50
CWE-758	4.00	4.00	4.00

high ranking gives a good indication that the overall event, contributes positively to raise software developers awareness about secure coding and secure coding guidelines. The two lowest-ranked questions (although still ranked positive) are X7 - understand the output of static application security testing tools, and X11 - challenges related to daily work in the company. It is also an expected result that X7 is not as high ranked as the other questions, since the Sifu platform is currently not designed to train software developers to use static code analysis tools. Nonetheless, the participants still conclude that the platform helps them to understand these tools better. Further research is needed in this direction. Lastly, among the eleven questions, X11 obtained the lowest agreement rank. This result is unexpected, as the challenges have been adapted to the participants' daily work. The authors think that this lower ranking might be related to either the introduction (Phase 1) or conclusion (Phase 3) of the challenge. Nevertheless, the agreement rate is still very positive.

Detailed analysis of survey S_2 and S_3

Figure 17 shows an analysis of Survey S_2 and S_3 in light of the different theories that were used to formulate the survey questions, namely: 1) Definition of Awareness by Hänsch and Benenson (2014), 2) Happy Developers by Graziotin et al. (2018) and 3) Work-related factors by the experience of the first author. The two work-related factors X7 and X8) that can contribute to the development of high quality software, according to the experience of the first author, are the following: the readiness of the software developer to understand the output of SAST tools, and knowing where to find further information about secure coding in the company. In the figure, the Awareness component is broken down into its three dimensions (cf. Hänsch and Benenson (2014)): Behaviour, Protection, and Perception. Also noted in the figure are the mappings of all the survey questions to each of the components (and each of Awareness dimensions). Finally, the numbers inside the ellipses represent the average agreement rate for the questions belonging to each component.

As we can observe from this figure, all the components have an average agreement with 4.13 points or higher, in a 5 point Likert scale. In terms of Awareness the ranking of the different dimensions is as follows: Protection (4.19), Behaviour (4.16), and Perception (4.15). Comparing between the three components, Awareness is first (with an overall average rating of 4.17, followed by Work-Related Factors (4.15) and Happiness (4.13). The fact that Work-related has a rank higher than Happiness is surprising. The authors' understanding is that the overall CSC event contributes to this result since the CSC event is primarily prepared to address the company's environment and needs.

Interpretation of the results in relation to research questions

All the results presented in this work indicate the suitability of the Sifu platform to raise the awareness of software developers in industry, about secure coding and secure coding guidelines.

The presented Sifu platform addresses RQ1, and RQ2 - automatic assessment of challenges is done using a

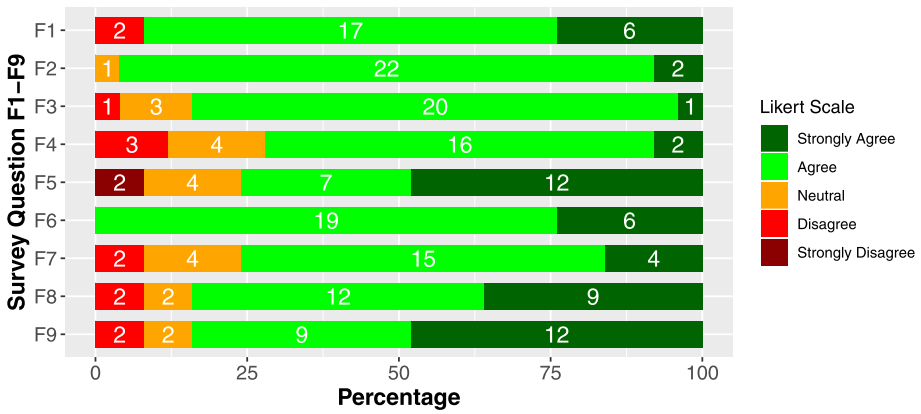

Fig. 15 Survey Results (Survey S_2)

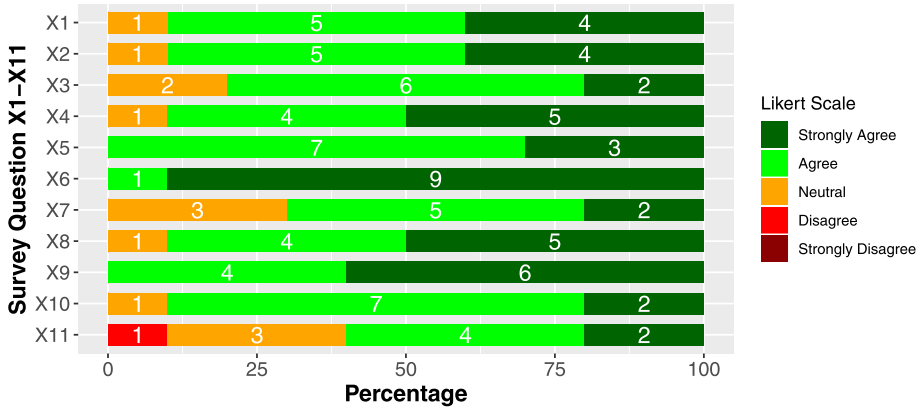

Fig. 16 Survey Results (Survey S_3)

combination of tools and methodologies, and software developers are aided in solving the challenges using an AI component based on laddering technique - the Intelligent Coach. Survey S_1 and S_2 were used to evaluate the Sifu platform and its suitability to raise secure coding awareness. All the results are encouraging towards validating the suitability statement.

Furthermore, after the initial event (Event nr. 1), we conducted further research (with Survey S_3) on the Sifu platform usage in real-world workshops held in the industry. Here, the last three CSC events' overall results using the Sifu platform (with 56 participants from the industry) show encouraging results towards RQ3.

Table 7 shows the top 10 quotes from the CSC participants, which was collected during the feedback phase, where all the participants were asked if they would like to share something about the event. The participants' qualitative feedback also confirms RQ3, in particular, that the CSC event with the Sifu Platform is fun and informative.

Additionally to this feedback, one participant (a professional software developer having a Bachelor in Computer Science) contacted the first author, after the event, with a request for information about further university studies on the topic of IT Security. The event caused such a good impression that the participant has decided to continue his studies, and pursue a Masters Degree in Computer Science in parallel to his work.

Threats to validity

In this work, we present a serious game called CyberSecurity Challenges and a code-entry challenge implemented in a platform that we call Sifu. The serious game and the Sifu platform are geared towards improving software developers' secure coding skills in the industry. To validate the Sifu platform's usefulness, the authors have gathered

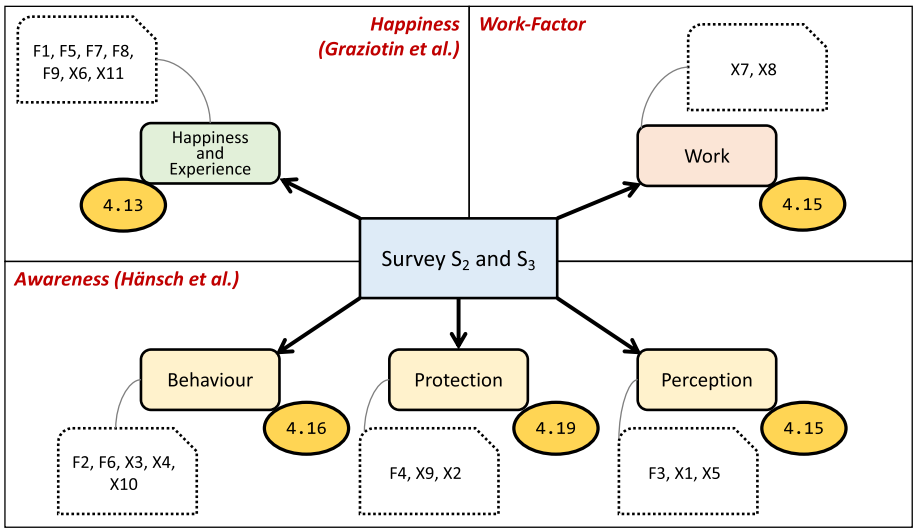

Fig. 17 Analysis of Survey S_2 and S_3

Table 7 Quotes from CSC Participants

Quote from Participant
I really enjoyed participating in the challenges. I am well excited in trying to crack the answers to the challenges
Enjoyed the challenges, the different topics and how competitive we got at the end
It was lots of fun. Questions inbetween were nice.
Enjoyed and lots of fun. I've learned many interesting things
Quite fun and nice to work, especially work in team
Enjoyed and learned very much
It was really funny and I leaned a lot
Funny and interesting day; learned a lot - hope I can remember and use in practice
Really liked and enjoyed the exercises
Enjoyable to try everything and very fun

feedback, in the form of two Surveys (S_1 and S_2), from participants of a trial experiment (Event nr. 1). The participants in this survey were composed of eight persons from academia and seven persons from the industry. Furthermore, three CSC events using the Sifu platform were performed with professional software developers from the industry. The total number of participants that took part in the three events was 56. In addition to the two surveys administered during the trial experiment (Event 1), an additional survey (S_3) was designed to capture the whole CSC event's usefulness when using the Sifu platform. During these last three events (Event 2, 3 and 4), all the surveys (S_1, S_2 and S_3) were administered to the participants. All the results give a good indication of the suitability of the Sifu platform, and the CSC event, as a means to raise awareness on secure coding and secure coding guidelines for software developers in the industry.

Possible sources of threat to the validity of the results, and conclusions presented are the following:

- *participant bias*: the participation in the different surveys was not mandatory; as such, we might consider that some possible negative answers have not been captured,
- *cultural differences*: participants in the survey included participants from different countries. We cannot exclude that differences in interpretation and language might affect the own experience in the CSC events, and
- *background and experience*: each participant has a different background and experience in the industry. These factors can lead to different perceptions of the proposed artifact.

To address the first concern, the feedback round in the CSC event was introduced - here, all the participants were individually asked if they would like to share anything. Although some participants have given suggestions to improve the Sifu platform further, there was a consensus that the platform is good among software developers. To address the second concern, during the CSC events, the two individual coaches monitor how the game is played, and help individual teams if there is any understanding problem, either with the goal, with the challenge, or about using the Sifu platform. Both coaches did not detect any problem, neither due to language barriers nor with cultural differences. Also, software developers are used to read, write, and program using the English language. Finally, with relation to the third concern, the CSC game is geared towards, and used by, software developers in the industry. Furthermore, the artifact is used in-house with success, where it is expected to have different groups of software developers, with different levels of expertise. What we have observed in practice was that the more advanced developers helped the other developers during the competition. The competition format gives the participants incentives to engage in discussions actively and search for a common solution. As such, the authors think that this can be a strength and not a CSC game's weakness.

The authors think that the presented data has a high degree of validity due to the reasons discussed. This conclusion matches the offline discussions that the coaches have had with the participants after the events.

Conclusions

Over the last years, the number of cybersecurity incidents on industrial control systems and cybersystems has been increasing. The root cause of some of these incidents can be traced back to poor software development practices. These poor software development practices are likely linked to software developers lack of awareness about secure coding. This is strongly supported by Patel et al. recent study (Patel 2020), which shows that more than 50% of software developers cannot spot vulnerabilities in source code. The lack of awareness is especially a problem for critical infrastructures, as the consequences of exploiting vulnerable code can range from simple interruption of service up to loss-of-life.

The industry sector is well regulated through IT standards, such as IEC 62.443 and ISO 27k. These standards mandate the implementation of a secure software development life cycle, as well as that software developers be adequately trained (and made aware) in developing secure code. Furthermore, these standards also instruct the industry's adoption of secure coding guidelines and secure coding policies.

One possible way to raise awareness of software developers, on secure coding and secure coding guidelines, is employing serious games. This game genre is a well-recognized means to achieve this goal, according to the

BSI-Grundschutz-Kompendium from the German Federal Office for Information Security. One such game, the CyberSecurity Challenges, which is based and adapted from the serious game genre of Capture-the-Flag, has been developed in the industry to address this issue - raising secure coding awareness of industrial software developers.

The present work extends the CyberSecurity Challenges utilizing a CyberSecurity Awareness Platform that the authors have named: Sifu. Previously existing CyberSecurity Challenges have focused on adapting existing open-source components to the industry, particularly in shifting the Challenges' focus to the defensive perspective. However, these challenges, due to their conception, lack inter action with the user. The Sifu platform, proposed in this work, breaks this limitation by providing a high degree of interactivity with the players, while still focusing on the defensive perspective. This platform's implementation is accomplished using two key ideas: automatic challenge assessment and intelligent hint generation.

When solving a challenge in the Sifu platform, the goal of the player is to fix or rewrite parts of the source code of a simple project, in such a way as to eliminate one or more known vulnerabilities, maintain the intended functionality, and follow secure coding guidelines. The automatic challenge assessment component makes use of existing open-source components to perform unit-testing, static, dynamic, and run-time security analyses of the project code, to determine if the player's solution is acceptable or not. One main advantage is that, due to the way the code submitted by the player is tested in the back-end, several solutions can be acceptable, i.e., the back-end does not compare the player's solution with a desired and fixed solution. Since the back-end needs to perform checks on untrusted code, it implements mechanisms that prevent cheating by the players, and mechanisms that do not allow them to attack the system back-end.

The proposed Sifu platform was evaluated through three surveys that targeted different aspects: 1) quality of the challenges, 2) Sifu platform, and 3) CSC event with Sifu platform. Our results show that the participants have fun using the platform, and find it an adequate means to raise awareness on secure coding best practices. Also, the Sifu platform's challenges have generally high ratings, indicating that software developers agree on the quality of the challenges. In terms of awareness, the Sifu platform has high feedback ratings. High feedback ratings were also consistently obtained for the work-related factors, as well as, for the contribution to developers happiness and good user experience.

With this work, we hope to positively impact both the industry and academia by laying out a novel methodology to raise secure coding awareness of software developers, that focus on defensive challenges, and is proving successful in the industry. The authors intend to make the Sifu platform available, in the future, after completing a necessary software clearing process.

In future work, the authors would like to perform a usability study of the platform and investigate ways to improve it. In particular, the authors have collected many ideas and suggestions on future improvements from the participants. We would also like to investigate which factors lead software developers to understand the consequences of exploiting vulnerable code, while participating in a CyberSecurity Challenges event. This investigation will allow us to further improve the challenge presentation. The authors would like to investigate additional ways to implement a more mature artificial engine for the intelligent coach, through systematic literature research. Finally, the intelligent coach engine's quality depends heavily on the quality and number of input sources. Towards this, the authors intend to investigate other sources of information that can be used to expand challenge assessment.

Abbreviations

AH: Analysis and hints; AI: Artificial intelligence; AUTOSAR: Automotive open system architecture; AVG: Average; BE: Behavior; BP: Best practices; BSI: Bundesamt für Sicherheit in der Informationstechnik; CERT: Computer emergency response team; CEC: Code-entry-challenge; CSC: Cybersecurity challenges; CPU: Central processing unit; CTF: Capture-the-flag; CWE: Common weakness enumeration; DAST: Dynamic application security testing; DHS: Department of homeland security; Gb: Gigabyte; HP: Happiness; ICS: Industrial control systems; IEC: International electrotechnical commission; ISO: International standard organization; IT: Information technology; JSON: Javascript object notation; MISRA: Motor industry software reliability association; NP: Number of players; OS: Open source; PCI/DSS: Payment card industry data security standard; PE: Perception; PR: Protection; RAM: Random access memory; RASP: Runtime application security testing; RQ: Research question; SAFECode: Software assurance forum for excellence in code; SAST: Static application security testing; SCG: Secure coding guideline; SEI-CERT: Software engineering institute - computer emergency response team; UK: United Kingdom; UT: Unit test; WK: Work; XML: Extensible markup language;

Acknowledgements

The authors would like to thank the survey participants for their useful and insightful discussions and for their participation in the survey. The authors would like thank Dr. Kristian Beckers and Thomas Diefenbach for their helpful, insightful, and constructive comments, discussions and suggestions. The authors would also like to thank Anmoal Porwal for his help in implementing part of the Sifu challenges during his working student position at Siemens AG.

Authors' contributions

This work was done by Tiago Gasiba under the supervision of Prof. Dr. Ulrike Lechner and Prof. Dr. Maria Pinto-Albuquerque. The main idea, implementation and evaluation in an industrial context was carried out by the first author. All the authors have actively contributed in steering the work and, in particular, in discussing adequate forms of evaluation of the platform, in terms of survey and their questions. All the authors made a significant contribution in the preparation, review and writing of this manuscript. All authors read and approved the final manuscript.

Availability of data and materials

The Sifu platform is available as an open source project, under the MIT license, and can be downloaded under the following link: https://gitlab.com/tgasiba/sifu.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Siemens AG Corporate Technology, Otto-Hahn-Rin 6, 81379 Munich, Bavaria, Germany. [2]Universität der Bundeswehr München, Munich, Germany. [3]Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, Lisbon, Portugal.

Received: 2 October 2020 Accepted: 20 October 2020
Published online: 15 December 2020

References

AbsInt (2020) RuleChecker. https://www.absint.com/rulechecker/. Accessed June 2020
AbsInt (2020) Astrée. https://www.absint.com/astree/index.htm. Accessed June 2020
Apextechservices (2017) NotPetya: World's First <DOLLAR/>10 Billion Malware. https://www.apextechservices.com/topics/articles/435235-notpetya-worlds-first-10-billion-malware.htm#. Accessed June 2020
AUTOSAR Consortium (2017) AUTOSAR:2017, Guidelines for the use of the C++14 language in critical and safety-related systems, Munich. https://www.autosar.org/
Bakan U, Bakan U (2018) Game-based learning studies in education journals: A systematic review of recent trends. Actualidades Pedagógicas 72:119–145. https://doi.org/10.19052/ap.5245
Baudin P, Bobot F, Bonichon R, et al (2020) Frama-C. https://frama-c.com/. Accessed June 2020
Black R (2004) Critical Testing Processes: Plan, Prepare, Perform, Perfect. Addison-Wesley Professional, Boston
Brisson A, Pereira G, Prada R, Paiva A, Louchart S, Suttie N, Lim T, Lopes RA, Bidarra R, Bellotti F, et al (2012) Artificial intelligence and personalization opportunities for serious games. In: Eighth Artificial Intelligence and Interactive Digital Entertainment Conference. Association for the Advancement of Artificial Intelligence, Worcester. pp 51–57. https://www.aaai.org/ocs/index.php/AIIDE/AIIDE12/paper/viewFile/5527/5764. Accessed Oct 2020
Bundesamt für Sicherheit in der Informationstechnik (2014) Baustein B 5.21 - Webanwendungen, Bonn, Germany. https://tinyurl.com/y25m2kxl. Accessed Feb 2020
Bundesamt für Sicherheit in der Informationstechnik (2019) BSI IT-Grundschutz-Kompendium - Umsetzungshinweise zum IT-Grundschutz-Kompendium 2019. https://tinyurl.com/BSI-Grundschutz-Kompendium-Ums. Accessed Feb 2020
Bundesamt für Sicherheit in der Informationstechnik (2020) BSI IT-Grundschutz-Kompendium. https://tinyurl.com/BSI-Grundschutz-Kompendium. Accessed Feb 2020
Carnegie Mellon University (2019) Secure Coding Standards. https://tinyurl.com/y29mwsyj. Accessed June 2020
Chapman P, Burket J, Brumley D (2014) Picoctf: A game-based computer security competition for high school students. In: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association, San Diego
Chung K (2020) CTFd : The Easiest Capture The Flag Framework. https://ctfd.io/. Accessed Mar 2019
Chung K (2017) Live lesson: Lowering the barriers to capture the flag administration and participation. In: 2017 USENIX Workshop on Advances in Security Education (ASE 17). USENIX Association, Vancouver. https://www.usenix.org/conference/ase17/workshop-program/presentation/chung. Accessed Mar 2019
Chung K, Cohen J (2014) Learning obstacles in the capture the flag model. In: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association, San Diego. https://www.usenix.org/conference/3gse14/summit-program/presentation/chung. Accessed Mar 2019
Crawley MJ (2012) The R Book. Wiley, United Kindom
Cullinane I, Huang C, Sharkey T, Moussavi S (2015) Cyber Security Education Through Gaming Cybersecurity Games Can Be Interactive, Fun, Educational and Engaging. J Comput Sci Coll 30(6):75–81
Davis A, Leek T, Zhivich M, Gwinnup K, Leonard W (2014a) The Fun and Future of CTF. In: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association, San Diego. pp 1–9
Davis A, Leek T, Zhivich M, Gwinnup K, Leonard W (2014b) The fun and future of CTF. In: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association, San Diego. https://www.usenix.org/conference/3gse14/summit-program/presentation/davis. Accessed Sep 2018
Department of Homeland Security (2020) ICS-CERT: Industrial Control Systems - Computer Emergency Response Team. https://us-cert.cisa.gov/ics. Accessed Jul 2020
Department of Homeland Security US-CERT (2020) Software Assurance. https://tinyurl.com/y6pr9v42. Accessed Jul 2020
Djaouti D, Alvarez J, Jessel J (2011) Classifying Serious Games: the G/P/S model. In: Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches. IGI Global, Hershey, Pennsylvania. pp 118–136. https://doi.org/10.4018/978-1-60960-495-0.ch006
Dobrovsky A, Borghoff UM, Hofmann M (2016) An approach to interactive deep reinforcement learning for serious games. In: 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom). IEEE, Wroclaw. pp 85–90
Dörner R, Göbel S, Effelsberg W, Wiemeyer J (2016) Serious Games: Foundations, Concepts and Practice. 1st edn.. Springer, Switzerland. https://doi.org/10.1007/978-3-319-40612-1
Drever E (1995) Using Semi-Structured Interviews in Small-Scale Research. A Teacher's Guide. Scottish Council For Research In Education, Edinburgh
ESBMC (2020) Efficient SMT-based Bounded Model Checker. http://www.esbmc.org/. Accessed June 2020
FaceBook (2020) FbInfer. https://fbinfer.com/. Accessed June 2020
Frey S, Rashid A, Anthonysamy P, Pinto-Albuquerque M, Naqvi SA (2019) The Good, the Bad and the Ugly: A Study of Security Decisions in a Cyber-Physical Systems Game. IEEE Trans Softw Eng 45(5):521–536
Gadelha MR, Monteiro FR, Morse J, Cordeiro LC, Fischer B, Nicole DA (2018) ESBMC 5.0: An industrial-strength C model checker. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ACM, New York. pp 888–891. https://doi.org/10.1145/3238147.3240481
Gasiba T, Beckers K, Suppan S, Rezabek F (2019) On the Requirements for Serious Games geared towards Software Developers in the Industry. In: Damian DE, Perini A, Lee S (eds). 2019 IEEE 27th International Requirements Engineering Conference (RE). IEEE, Jeju Island. https://ieeexplore.ieee.org/xpl/conhome/8910334/proceeding
Gasiba T, Lechner U, Cuellar J, Zouitni A (2020a) Ranking Secure Coding Guidelines for Software Developer Awareness Training in the Industry. In: First International Computer Programming Education Conference (ICPEC 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Vila do Conde, Porto, Portugal. Virtual Conference
Gasiba T, Lechner U, Pinto-Albuquerque M, Zouitni A (2020b) Design of Secure Coding Challenges for Cybersecurity Education in the Industry. In: International Conference on the Quality of Information and Communications Technology. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Vila do Conde, Porto, Portugal. pp 223–237
Gimpel (2020) PcLint. https://www.gimpel.com/. Accessed June 2020
Google (2020a) American Fuzzy Lop. https://github.com/google/AFL. Accessed June 2020
Google (2020b) Address Sanitizer. https://github.com/google/sanitizers. Accessed June 2020
Google (2020c) Leak Sanitizer. https://github.com/google/sanitizers/wiki/AddressSanitizerLeakSanitizer. Accessed June 2020
Google (2020d) Thread Sanitizer. https://github.com/google/sanitizers. Accessed June 2020
Goseva-Popstojanova K, Perhinschi A (2015) On the capability of static code analysis to detect security vulnerabilities. Inf Softw Technol 68:18–33
Graziotin D, Fagerholm F, Wang X, Abrahamsson P (2018) What happens when software developers are (un)happy. J Syst Softw 140:32–47
Groves RM, Fowler F, Couper M, Lepkowski J, Singer E (2009) Survey Methodology. 2nd edn. Wiley, New Jersey
Hänsch N, Benenson Z (2014) Specifying IT security awareness. In: 25th International Workshop on Database and Expert Systems Applications, Munich, Germany. pp 326–330. https://doi.org/10.1109/DEXA.2014.71
Harrell MC, Bradley MA (2009) Data collection methods: semi-structured interviews and focus groups. annotated. RAND, 2009. https://www.rand.org/pubs/technical_reports/TR718.html

Hendrix M, Al-Sherbaz A, Bloom V (2016) Game based cyber security training: are serious games suitable for cyber security training? Int J Serious Games 3. https://doi.org/10.17083/ijsg.v3i1.107
Hulin P, Davis A, Sridhar R, Fasano A, Gallagher C, Sedlacek A, Leek T, Dolan-Gavitt B (2017) Autoctf: Creating diverse pwnables via automated bug injection. In: 11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association, Vancouver
IEC 62443-4-1 (2018) Security for industrial automation and control systems - part 4-1: Secure product development lifecycle requirements. Standard, International Electrotechnical Commission, Geneva. https://webstore.iec.ch/publication/33615. Accessed Jun 2020
IEEE Spectrum (2019) The Top Programming Languages 2018. https://tinyurl.com/y75qj2ea. Accessed June 2020
ISO 27001 (2013) Information technology – Security techniques – Information security management systems – Requirements Standard, International Standard Organization, Geneva, CH, Geneva. https://www.iso.org/standard/54534.html. Accessed Jun 2020
JMMV (2020) Automated Test Framework. https://github.com/jmmv/atf. Accessed June 2020
JMMV (2020) Kyua - A Testing Framework for Infrastructure Software. https://github.com/jmmv/kyua. Accessed June 2020
Joshi A, Kale S, Chandel S, Pal D (2015) Likert scale: Explored and explained. Br J Appl Sci Technol 7:396–403. https://doi.org/10.9734/BJAST/2015/14975
Kässtner D, Hahn S, Herter J, Karos T (2020) Undecidable rules and how to live with them. https://www.embedded-world.de/en/events/vortrag/undecidable-rules-and-how-to-live-withthem/742776. Accessed Jun 2020
Lattner C (2018) clang: a C language family frontend for LLVM. https://clang.llvm.org/index.html. Accessed June 2020
Leune K, Petrilli Jr S (2017) Using capture-the-flag to enhance the effectiveness of cybersecurity education. In: Proceedings of the 18th Annual Conference on Information Technology Education. Association for Computing Machinery, Rochester. pp 47–52
Marjamäki D (2017) CppCheck. http://cppcheck.sourceforge.net/. Accessed June 2020
Mead N, Allen J, Barnum S, Ellison R, McGraw G (2004) Software Security Engineering: a Guide for Project Managers. Addison-Wesley Professional, Boston
MetricsGrimoire (2020) CMetrics. https://github.com/MetricsGrimoire/CMetrics. Accessed June 2020
Mirkovic J, Peterson PA (2014) Class capture-the-flag exercises. In: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association, San Diego
Misra C (2012) Guidelines for the use of the C language in critical systems, Nuneaton, Warwickshire, UK. https://www.misra.org.uk/MISRAHome/MISRAC2012/tabid/196/Default.aspx. Accessed Jun 2020
MITRE (2020) CWE 14: Compiler Removal of Code to Clear Buffers. https://cwe.mitre.org/data/definitions/14.html. Accessed Jun 2020
MITRE (2020) CWE 77: Improper Neutralization of Special Elements used in a Command ('Command Injection'). https://cwe.mitre.org/data/definitions/77.html. Accessed Jun 2020
MITRE (2020) CWE-121: Stack-based Buffer Overflow. https://cwe.mitre.org/data/definitions/121.html. Accessed Jun 2020
MITRE (2020) CWE-242: Use of Inherently Dangerous Function. https://cwe.mitre.org/data/definitions/242.html. Accessed Jun 2020
MITRE (2020) CWE 338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG). https://cwe.mitre.org/data/definitions/338.html. Accessed Jun 2020
MITRE (2020) CWE 676: Use of Potentially Dangerous Function. https://cwe.mitre.org/data/definitions/676.html. Accessed Jun 2020
MITRE (2020) CWE 758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior). https://cwe.mitre.org/data/definitions/758.html. Accessed Jun 2020
Misra (2012) MISRA-C:2012 Amendment 1, Additional security guidelines for MISRA C:2012, Nuneaton, Warwickshire, UK. https://www.misra.org.uk/MISRAHome/MISRAC2012/tabid/196/Default.aspx, Accessed June 2020
Nakamura J, Csikszentmihalyi M (2014) The concept of flow. In: Flow and the Foundations of Positive Psychology. Springer, Dordrecht. pp 239–263
NASA-SW-VnV (2020) Ikos. https://github.com/nasa-sw-vnv/ikos. Accessed June 2020
Patel S (2020) 2019 Global Developer Report: DevSecOps finds security roadblocks divide teams. https://about.gitlab.com/blog/2019/07/15/global-developer-report/. Accessed Aug 2020
PCI DSS (2015) Requirements and security assessment procedures. https://www.pcisecuritystandards.org/. Accessed Jun 2020
R Core Team (2019) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna. https://www.R-project.org/. Accessed Jan 2020
R2C (2020) SemGrep. https://semgrep.dev/. Accessed June 2020
Rieb A (2018) IT-Sicherheit: Cyberabwehr mit hohem Spaßfaktor. In: Kma - Das Gesundheitswirtschaftsmagazin, vol. 23. Georg Thieme Verlag KG, Stuttgart. pp 66–69. https://www.thieme-connect.com/products/ejournals/abstract/10.1055/s-0036-1595355. Accessed June 2020
Rieb A, Gurschler T, Lechner U (2017) A gamified approach to explore techniques of neutralization of threat actors in cybercrime. In: GDPR & ePrivacy: APF 2017 - Proceedings of the 5th ENISA Annual Privacy Forum. Springer, Cham. pp 87–103
Rietz T, Maedche A (2019) LadderBot: A Requirements Self-Elicitation System. In: 2019 IEEE 27th International Requirements Engineering Conference (RE). IEEE Computer Society, Jeju Island. pp 357–362
Schneier B (2020) Software Developers and Security. https://www.schneier.com/blog/archives/2019/07/software_develo.html. Accessed Aug 2020
Siemens AG (2020) Siemens Charter of Trust. https://www.charteroftrust.com/. Accessed Feb 2019
Simões A, Queirós R (2020) On the nature of programming exercises. In: ICPEC - First International Computer Programming Education Conference, ICPEC, vol. 81. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Vila do Conde. pp 251–259. Virtual Conference
Software Assurance Forum for Excellence in Code (2018) SAFECode - Fundamental Practices for Secure Software Development - Essential Elements of a Secure Development Life-cycle Program, 3rd Ed. https://tinyurl.com/y44etrs7. Accessed Sep 2019
SonarSource (2020) SonarQube. https://www.sonarqube.org/. Accessed June 2020
Sorace S, Quercia E, La Mattina E, Patrikakis CZ, Bacon L, Loukas G, Mackinnon L (2018) Serious Games: An Attractive Approach to Improve Awareness(Leventakis G, Haberfeld MR, eds.). Springer, Cham
Stallman RM (2002) GNU compiler collection internals. Free Softw Found
Švábenskỳ V, Vykopal J, Cermak M, Laštovička M (2018) Enhancing cybersecurity skills by creating serious games. In: Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education. pp 194–199. https://wsiw2018.l3s.uni-hannover.de/papers/wsiw2018-Votipka.pdf. Accessed Oct 2020
Tabassum M, Watson S, Chu B, Lipford HR (2018) Evaluating two methods for integrating secure programming education. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. Association for Computing Machinery, Baltimore. pp 390–395. https://doi.org/10.1145/3159450.3159511
Tecent (2020) TScanCode. https://github.com/Tencent/TscanCode. Accessed June 2020
The Clang Team (2020) Clang-Tidy. https://clang.llvm.org/extra/clang-tidy/. Accessed June 2020
Valgrind Developers (2020) Helgrind. https://www.valgrind.org/docs/manual/hg-manual.html. Accessed June 2020
Valgrind Developers (2010) Valgrind. https://valgrind.org/. Accessed June 2020
Vasconcelos P, Ribeiro RP (2020) Using property-based testing to generate feedback for C programming exercises. In: ICPEC - First International Computer Programming Education Conference, ICPEC, vol. 81. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Vila do Conde, Porto, Portugal. pp 285–294. Virtual Conference
Votipka D, Eastes B, Mazurek M (2018a) Toward a field study on the impact of hacking competitions on secure development. In: The 4th Workshop on Security Information Workers Baltimore Marriott Waterfront, Baltimore
Votipka D, Mazurek ML, Hu H, Eastes B (2018b) Toward a Field Study on the Impact of Hacking Competitions on Secure Development. In: Workshop on Security Information Workers (WSIW). Online, Baltimore. Marriott Waterfront. https://wsiw2018.l3s.unihannover.de/papers/wsiw2018-Votipka.pdf. Accessed Oct 2020
Wagner S, Mendez D, Felderer M, Graziotin D, Kalinowski M (2020) Challenges in survey research. In: Michael Felderer GHT (ed). Contemporary Empirical Methods in Software Engineering. Springer, ArXiv. pp 1–34
Wheeler D (2013) FlawFinder. https://dwheeler.com/flawfinder/. Accessed June 2020

WhiteSource (2019) What are the Most Secure Programming Languages? https://www.whitesourcesoftware.com/most-secure-programming-languages/. Accessed June 2020
Wireghoul (2020) Graudit. https://github.com/wireghoul/graudit. Accessed June 2020

Publisher's Note