Is Secure Coding Education in the Industry Needed? An Investigation Through a Large Scale Survey

Tiago Espinha Gasiba
Siemens AG
Munich, Germany
tiago.gasiba@siemens.com

Ulrike Lechner
Universität der Bundeswehr München
Munich, Germany
ulrike.lechner@unibw.de

Maria Pinto-Albuquerque
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL
Lisboa, Portugal
maria.albuquerque@iscte-iul.pt

Daniel Mendez
Blekinge Institute of Technology and fortiss GmbH
Karlskrona, Sweden
daniel.mendez@bth.se

***Abstract*—The Department of Homeland Security in the United States estimates that 90% of software vulnerabilities can be traced back to defects in design and software coding. The financial impact of these vulnerabilities has been shown to exceed 380 million USD in industrial control systems alone. Since software developers write software, they also introduce these vulnerabilities into the source code. However, secure coding guidelines exist to prevent software developers from writing vulnerable code. This study focuses on the human factor, the software developer, and secure coding, in particular secure coding guidelines. We want to understand the software developers' awareness and compliance to secure coding guidelines and why, if at all, they aren't compliant or aware. We base our results on a large-scale survey on secure coding guidelines, with more than 190 industrial software developers. Our work's main contribution motivates the need to educate industrial software developers on secure coding guidelines, and it gives a list of fifteen actionable items to be used by practitioners in the industry. We also make our raw data openly available for further research.**

***Index Terms*—education, training, industry, secure coding guidelines, software developers, awareness, survey**

I. Introduction

According to a Kaspersky [1] report, businesses spent an average of 380 million USD in 2017 to recover and deal with the consequences of Industrial Control Systems (ICS) incidents, and this value is still increasing. Gartner's 2019 report predicts that the financial impact of attacks on Cyberphysical Systems will exceed 50 billion USD in 2023. The United States Department of Homeland Security estimates that the vast majority of security incidents can be attributed to defects in software design and code [2].

To deliver secure software-based products and services, we must consider security while producing software. To become certified and able to conduct business in the critical infrastructure sector, companies must comply with several standards. Among these standards, IT Security standards such as the ISO 27k [3] and IEC 62.443 [4] mandate, among others, the establishment of a secure software development lifecycle (S-SDLC). The S-SDLC includes the usage of secure coding guidelines (SCG) and the checking of code quality (ISO 25k [5]) against these guidelines.

Secure coding, secure software development, and secure coding guidelines are no easy subjects. Some of the vastly known and adopted SCG include Carnegie Mellon's Software Engineering Institute C, C++, and Java secure coding guidelines standars [6] (also known as SEI-CERT), the Motor Industry Software Reliability Association standard (MISRA) [7], [8], and the Open Web Application Security Project Top 10 (OWASP standard) [9]. However, SCG do not exist for all existing programming languages. In addition to SCG, to address the importance of secure code and the need to develop secure products, several companies united to form the SAFEcode [10]. This alliance promotes secure coding and industrial secure coding best practices.

Automatic tools such as Static Application Security Testing (SAST) [11] can be used to automate and improve code quality. These tools scan the code basis for existing vulnerabilities, which must be fixed by software developers. However, previous research shows that their reliability is not good enough [12], in particular they exhibit a large amount of false positives and false negatives. Also, these tools cannot automatically fix the code – software developers must do this.

In this work, we focus on the human factor, i.e. the software developer. We justify this focus since it is the software developer who writes the code, who interprets the output of SAST tools, and who will ultimately be the person that introduces software weaknesses into the code basis. It will be the software developer as well that will have to correct the vulnerabilities in code. This study is embedded in our investigation on the usage of serious games as a means to raise secure coding awareness of software developers in the industry [13]–[21]. Our primary motivation to conduct the present work is to motivate awareness training by answering the question "is secure coding education in the industry needed?". Our study focuses particularly on the education of secure coding guidelines. We approach this question by looking at the perspective of software developers' compliance to secure coding guidelines.

Due to a lack of previous work exploring the relationship between secure coding guidelines and software developers' intention to comply with them, we have developed a survey to address this issue. Our previous publication details the overall research method underlying this survey and is available in [14]. However, this previous publication focuses on the survey creation and only presents limited results from the survey pilot, i.e. it does not present any results of a large-scale

deployment of the survey. This work at hands closes this gap and presents an extensive analysis of a large-scale deployment. We base our results on 194 answers from participants working in different industries, collected over a period of seven months. Our analysis of these results addresses the following research questions:

RQ1: Which factors lead industrial software developers to comply with or ignore secure coding guidelines?

RQ2: To what degree are software developers aware of secure coding guidelines?

RQ3: To what extent is secure coding education in the industry needed?

Through the large-scale survey, our contribution to scientific knowledge comprises:

1) openly available data from a large-scale survey, for other researchers to explore,
2) the presentation and interpretation of results from the analysis of the survey, and
3) a list of actionable items for practitioners and industrial cybersecurity educators.

This paper is organized as follows. Section II, briefly discusses previous and related work. Section III gives a very brief overview of the survey and its theoretical constructs. In section IV, we present a comprehensive overview of the most important results from the analysis of the survey, derive actionable items, and discuss the threats to the validity. This section, which constitutes the core of the paper, provides herein our main contribution. Finally, section V concludes this paper with an overview of the study, and an outline of further work.

II. Related Work

Based on a large-scale study by Patel et al. [22], Bruce Schneier, a well-known security researcher, has stated that less than 50% of software developers can spot security vulnerabilities in software [23]. Also, an estimation by the United States Department of Homeland Security, about 90% of the *reported security incidents result from exploits against defects in the design or code of software* [2]. Adding to these facts, software is becoming more complex and larger: a recent study by Sourcegraph [24], with more than 500 software developers, shows that more than 80% of software developers are nowadays dealing with 20 times more code than ten years before.

An additional motivating factor for our work is Fisher et al. [25], which shows that typical online platforms that software developers use to clarify development questions can be considered harmful. The reason for not being a good source of information is that the answers present in these platforms are not curated in secure coding correctness. Their work indicates that severe problems can arise if software developers use these references and are not aware of secure coding practices. Furthermore, Acar et al. [26] extensively analysed existing online resources that software developers can access to search about secure programming issues. They discovered that these platforms provide low-quality information in terms of cybersecurity. In particular, they found outdated information, wrong information, and no concrete examples or exercises.

While many studies focus on several different aspects of secure software development, very few empirical results exist on why software developers do not comply with secure coding practices. In particular, to the best of our knowledge, we have found no previous study addressing the aspects that lead industrial software developers to comply or not comply with secure coding guidelines in their daily work. In a recent study Assal et al. [27] analyzed how software developers are influenced and influence the secure coding processes. They concluded that software developers are *not the weakest link*, and are very motivated towards software security. However, they did not cover the reasons why this is so. In 2011, Xie et al. [28] interviewed 15 senior professional software developers in the industry with an average of 12 years of experience. Their study shows a disconnect between software security concepts and the knowledge that the participants have in their jobs. However, this study also does not focus on compliance to secure coding guidelines.

To address this issue, we have formally developed a survey [14] to investigate software developers' compliance to secure coding guidelines. This survey is based on the adaptation of four distinct theories to the software developer context: IT Security Policy Compliance theory (PC), IT Security Neutralization theory (NT), Security-Related Stress theory (SRS), and IT Security Awareness (AW). The work from Bulgurcu et al. [29] and Moody et al. [30]) synthesizes the current research on IT security policy compliance. Their work details the possible reasons that serve as factors for individuals to comply with IT security policies. Their constructs include, among others, the intention to comply and the knowledge of the policies. Neutralization Theory is is discussed in [31], by Siponen et al., who address the possible reasons why subjects might find reasons to disregard IT security policies. Their constructs include, among others, the metaphor of the ledger, denial of injury, denial of responsibility, and appeal to higher loyalties. D'Arcy et al. in [32], discuss Security-Related Stress theory which uses coping theory to explore stress as a cause of deliberate IT security policy violations. Their constructs include, among others, the lack of understanding, higher workload, and constant changes. Finally, Hänsch et al. provide a literature review on IT-security awareness [33]. Their conceptualization of IT Security Awareness comprises three distinct constructs: Perception, Protection, and Behavior. Perception relates to knowing existing threats, Protection relates to knowing existing mechanisms, and Behavior relates to actual behavior. Finally, in this work, we use the recent results from WhiteSource [34], which present the top three vulnerabilities of the C, C++, Java, and Python programming languages.

III. Survey on Secure Coding Guidelines

To investigate the possible reasons why vulnerabilities end up in final products, we have created a survey, described

in [14], that focuses on industrial software developers. This survey is based on the four established theories: policy compliance (PC, [29], [30]), neutralization theory (NT, [31]), security-related stress (SRS, [32]), and awareness (AW, [35]). Furthermore, the survey contains additional questions based on the industry experience by the first author. These questions are grouped by company background (CBG) and participant background knowledge (BGK). Table I shows the questions present in the survey, along with the different theories and constructs in which it is based.

The questionnaire comprises the following four sections: 1) demographic data, 2) secure coding awareness, 3) secure coding compliance, and 4) deterrents to compliance. The first part of the questionnaire includes general demographic questions on work experience, previous training on secure coding, the primary programming language used at work, used secure coding processes in the company, and software development method. The second section of the questionnaire deals with awareness for secure coding. This part is individualized according to the primary programming language selected in the first section. In this section, the participant is asked four questions related to high-impact vulnerabilities, according to [34]. These vulnerabilities are presented by the corresponding CWE [36] description and number, The four questions in this group correspond to Per1, Prot1, Be1 and BgK45, as shown in Table I. The answers to these (and only these) questions are based on a 3-point Likert scale: Yes, Uncertain and No. The third section of the questionnaire presents questions to measure the intent to comply to secure coding guidelines. These are the questions marked with PC in Table I. Finally, the fourth section contains questions about the factors that influence compliance with secure coding guidelines. These questions are based on neutralization theory and security-related stress, which are marked with NT and SRS in Table I, respectively.

The answers to the questions are based on a 5-point Likert scale, which include the following: strongly disagree (SD), disagree (D), neutral (N), agree (A) and strongly agree (SA). The following mapping is used in our results: SD↔1, D↔2, N↔3, S↔4, SA↔5. In the previous publication, we presented the rationale and details on how the survey was scientifically constructed. However, the preliminary results from the survey pilot available in this previous work only included a minimal subset of the survey questions, leading to very limited conclusions. For more details on the survey' questions and the design of the survey, we refer the reader to [14].

A. A Large Scale Survey

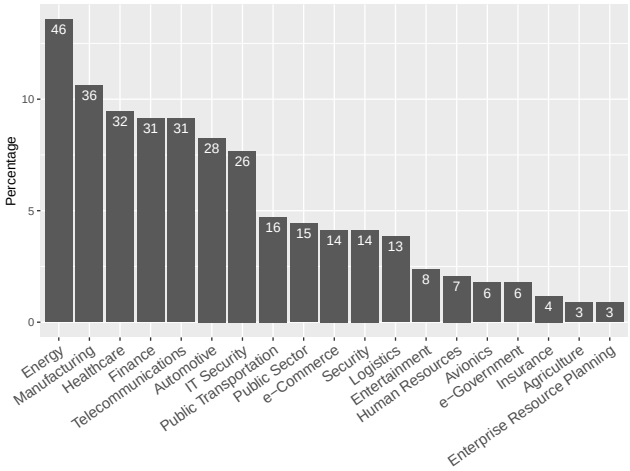
A large scale deployment of the survey was performed between March and September of 2020, resulting in a total running time of seven months. The survey was announced through several different channels, in particular:

1) **Professional Contacts**: Linked-In, direct contacts by the authors at several different companies, Münchener Sicherheitsnetzwerk (Munich Security Network Forum)
2) **Social Media**: Twitter, Facebook, Reddit
3) **Other**: University contacts, survey exchange platform (SurveySwap), advertisement in university website

The survey was constructed using the open-source survey platform LimeSurvey [37] Version 3.17.0+190402 and deployed in Amazon Web Services. At the beginning of the survey, it was clearly stated: the purpose of the research, contact details, and the mandatory requirement that the participant must be a software developer from the industry. Over the seven months, the survey was accessed 363 times resulting in 196 complete answers. Two answers were rejected due to irregularities found in the collected data. The full set of captured data is available under the following link [38]. All the data was anonymously collected; however, a cookie was activated to prevent participants from submitting twice their answers.

Figure 1 shows the different background industries captured by this data set. The demographics in terms of participants' programming languages are the following: C++ 50 (26%), Java 38 (20%), Python 37 (19%), Other 36 (18%), C 33 (17%). The survey was anonymous, and geographical, education and gender aspects were not captured.

Fig. 1. Survey demographics in terms of industry



In the next section, an extensive discussion of the results of the survey data analysis will be presented.

IV. Results

This section presents the survey results, categorized by the different theories in which it is based: CBG, BGK, PC, NT, SRS, and AW. The section concludes with the main practical take-aways from the analysis and discusses the threats to the results' validity.

A. Company Background

Table II shows the results for the company background constructs CBg1, CBg2, and CBg8. From these results, we observe that, in general, *compliance to secure coding guidelines is not being checked in the industry*, and that software developers are not sure about the secure software development

TABLE I
SURVEY QUESTIONS, THEORIES AND CONSTRUCTS

Theory	Ref.	Construct	Survey Question
CBG	—	*CBg1*	In your company compliance to secure code guidelines is being checked in projects you work in
		CBg2	You know the secure software development lifecycle in your company
		CBg3	To which extent do you work with the _____ secure coding standard?
		CBg4	Could you explain why you use secure coding guidelines when writing code for the product you currently develop?
		CBg5	Could you tell us why you do not use secure coding guidelines?
		CBg6	Why is compliance to secure coding guidelines not actively being checked in the projects you work in?
		CBg7	How is the compliance to secure coding guidelines checked in your current project?
		CBg8	In your company you use a well established secure software development life-cycle
BGK	—	*BgK1*	Compliance to secure coding guidelines is an important part of the development of company's products
		BgK2	Which of the following secure coding standards and best practices do you know?
		BgK3	You are aware of negative consequences resulting from exploiting vulnerabilities in the products you work for
		BgK4	What other weaknesses do you pay attention to in developing software for the product you currently work for?
		*BgK5**	You know about this weakness
PC	[29]	ISPA	You know that your company has a policy that mandates the usage of secure coding guidelines in software development
		ITC	You intend to always comply with secure coding guidelines
		GISA	You are aware of the existing security threats to the products of your company
		SE-C1	In your opinion, to write secure code, you have the necessary skills
		SE-C2	In your opinion, to write secure code, you have the necessary knowledge
		SE-C3	In your opinion, to write secure code, you have the necessary competency
	[30]	FacCond5	Support is available if you experience difficulties in complying with secure coding guidelines
		RespCost4	Secure coding guidelines make the task of writing software more difficult
	—	*PC-Conf*	Complying to SCG makes you feel more confident about the security of the code that you write
		PC-NT	In your opinion, to write secure code, you have the necessary time
		PC-NR	In your opinion, to write secure code, you have the necessary resources
		PC-NF	In your opinion, to write secure code, you have the necessary freedom
NT	[31]	N-DON3	It is OK to disregard secure coding guidelines when this means that you deliver your work-packages faster
		N-ATHL1	It is OK to disregard secure coding guidelines when you would otherwise not get your job done
		N-DOI1	It is OK to disregard secure coding guidelines when this would result in no harm to the customer
		N-DOI2	It is OK to disregard secure coding guidelines if no damage is done to the company you work for
		N-DOR3	It is OK to disregard secure coding guidelines if you do not understand them
		N-COC1	It is not as wrong to ignore secure coding guidelines that are not reasonable
		N-COC2	It is not as wrong to ignore secure coding guidelines that require too much time to comply with
		N-MOTL1	You feel that your general adherence to secure coding guidelines compensates for occasionally ignoring them
	—	*NT-MArc*	It is OK to disregard secure coding practices when this would lead to major architectural changes
		NT-CH	It is OK to disregard secure coding guidelines when this means that it makes your company's customers happy
		NT-SC	It is OK to disregard secure coding guidelines if the software is not safety critical
SRS	[32]	CX2	You find that new employees often know more about secure coding than you do
		CX4	You often find it difficult to understand your organization's security coding guidelines
		OL1	Complying to secure coding guidelines forces you to do more work than you can handle
		OL4	You are forced to change your work habits to adapt to your organization's secure coding guidelines
		UC1	There are constant changes in secure coding guidelines your organization
		UC4	There are constant changes in security-related technologies in your organization
AW	[33]	*Per1**	You can recognize code that contains this weakness
		*Be1**	You know how to write code that does not contain this weakness
		*Prot1**	You understand the possible consequences that can result from exploiting this weakness

RQ.: Research Question, **CBG**: Company Background, **BGK**: Participant Background Knowledge, **PC**: Policy Compliance Theory, **NT**: Neutralization Theory, **SRS**: Security-Related-Stress Theory, **AW**: Awareness, Note: constructs marked with * are specific for different programming languages

TABLE II
COMPANY BACKGROUND (CBG) AND BACKGROUND KNOWLEDGE(BGK)

	CBG			BGK	
	CBg1	CBg2	CBg8	BgK1	BgK3
Average	2,57	2,93	2,42	2,36	3,87
Standard Deviation	1,29	1,14	1,22	1,28	0,90

life-cycle (S-SDLC) used in their company. This observation is corroborated by the CBg8 results.

Table III shows the results for CBg4, CBg5, and CBg6. Here we observe that about 50% of the participants (out of the 23 that mentioned security being or not a requirement) claim that in their industry, implementation of security during product development is a requirement, while the other half state that this is not the case. In terms of factors why SCG are not used (CBg5), we found the following important factors: 1) lack of awareness (focus on products and not on security, and limited knowledge), 2) relying on SAST tools, 3) because the participants had no previous experience with issues, and 4) limited or lack of management commitment, and resources devoted to security. The main factor not to check compliance to SCG (CBg6) is the fact that SCG is not used, and the industry focuses on products, not on security. Another important factor was the (perceived) lack of automatic tools to assist in the compliance checks and especially the lack of awareness.

Figure 2 shows the results of CBg3: to which extent are standard secure coding guidelines used in the industry. For all the secure coding standards that the survey has covered, all the results show that they are not really used in practice.

TABLE III
RESULTS FOR COMPANY BACKGROUND: CBG4, CBG5, AND CBG6

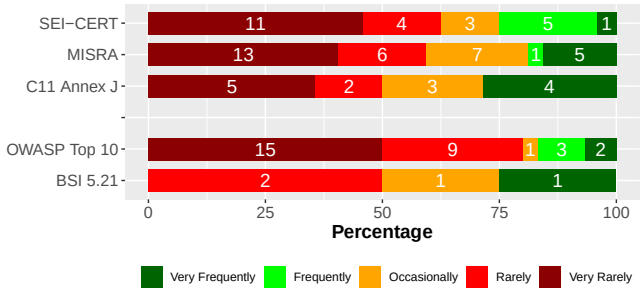
CBg4	
Why use secure coding guidelines?	No.
Security is a requirement	11
Because of compliance checks	5
Makes code resistant to attaks	4
Code is safe and reliable	3
Due to quality and data protection	3
Imposed by project quality gates	2
Ensure code quality	2
It's software development best practices	1
Comfortable with security	1
To avoid bugs	1
To reduce security risks	1

Legend:
SCG: Secure Coding Guidelines

CBg5	
Why not use secure coding guidelines?	No.
Not a requirement	12
Focus on products, not security	9
Limited knowledge	4
Takes too much time	2
Rely on SAST tools	2
Due to real-time constraints	2
Software deployed in secure environment	2
Customers do not "see" the feature	2
Security is added afterwards	2
Due to usage of proprietary software tools	1
Old code-base (e.g. >10 years)	1
Use open-source software	1
Cost saving reasons	1
Until now we had no issues	1
Time pressure	1

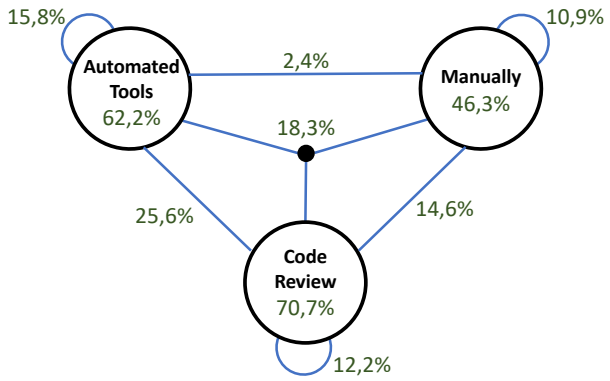
CBg6	
Why is compliance to SCG not being checked?	No.
Not using secure coding guidelines	4
Focus on products, not security	4
Not required by customer	3
Lack of resources	2
Products are not safety-critical	2
Lack of automatic tools to assist in compliance checks	1
Not enough higher management commitment	1
Nobody in the projects thinks about security	1
Lack of time	1
Small company	1
Security is an add-on	1
Cost saving	1
Not in our software development process	1
Lack of awareness	1
Security is not understood by software developers	1

Fig. 2. CBg3: To which extent use secure coding standard



For the participants who answered that SCG are checked in their company, fig. 3 shows how they are being checked. This figure shows that 70.7% check SCG during code review, 62,2% using automated tools, and 46.3% by a manual process. This figure also shows that 15.8% of the checks are done using automated tools exclusively, 12.2% using code review exclusively, and 10.9% through a manual process. Employing two different methods (Automated Tools and Code Review) lead to 25.6% of the results. About 18.3% claim that the three methods are used simultaneously.

Fig. 3. CBg7: How are secure coding guidelines checked?



B. Participant Background Knowledge

Table II shows that compliance to secure coding guidelines (BgK1) is not considered an essential part of the development of products, with an average agreement of 2.36. However, the survey participants have indicated to be aware (3.87 average agreement) of the negative consequences of exploiting software vulnerabilities (BgK3). We attribute this observation to the large amount of advertisement, e.g., social media, on these negative consequences.

Fig. 4. BgK2: Knowledge of SCG standard

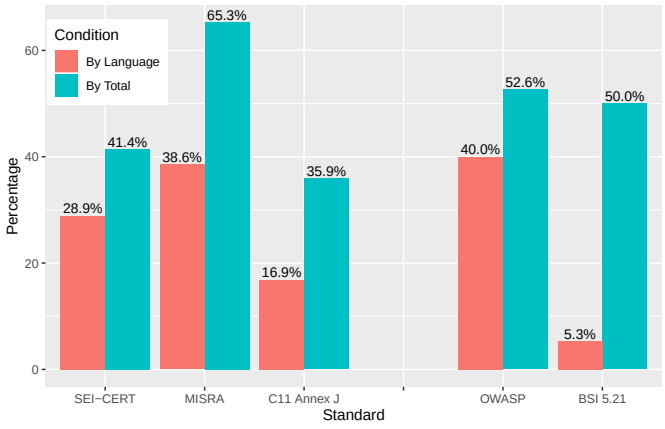


Figure 4 shows the extent to which survey participants know the different secure coding standards. The blue bars represent the ”percentage of the total survey population that knows the given SCG standard”. The red bars show the ”percentage of the population that should know the standard, given their chosen programming language”. For the SEI-CERT standard, this corresponds to the population who answered C or C++ as a programming language. For the MISRA and C11 Annex J the results correspond to C programmers. The OWASP and BSI standards capture Java, Python and programmers of Other languages.

General knowledge about SCG is low (below 65.3%). Comparing the blue to the red bars, these results show that

TABLE IV
RESULTS ON PC, NT AND SRS VS INDUSTRY, PROGRAMMING LANGUAGE AND WORK EXPERIENCE

		Measure	Policy Compliance												Neutralization Theory										Security-Related Stress					
			GISA	ITC	ISPA	Fac Cond5	PC-Conf	PC-NT	PC-NR	PC-NF	SE-C1	SE-C2	SE-C3	Resp Cost4	N-DOI1	N-DOI2	N-DON3	N-ATHL1	N-DOR3	N-COC1	N-COC2	N-MTOL1	NT-MArc	NT-CH	NT-SC	CX2	CX4	OL1	OL4	UC1	UC4
Industry	Finance	*AA*	3,7	3,8	3,5	3,7	3,7	3,3	3,3	3,6	3,7	3,5	3,5	3,4	3,1	2,9	2,3	2,3	2,4	2,9	2,5	2,7	2,8	2,6	2,9	2,6	2,8	3,0	3,0	2,7	3,3
		σ	0,7	0,8	1,2	0,9	0,7	0,8	0,8	0,7	0,7	0,8	0,8	0,9	1,0	0,9	1,1	1,0	0,9	1,1	0,8	0,9	0,9	1,0	1,0	0,8	0,9	0,8	0,8	0,9	0,9
	Healthcare	*AA*	3,7	3,6	3,5	3,5	3,5	3,3	3,2	3,5	3,8	3,6	3,6	3,2	2,8	2,4	2,1	2,5	2,1	3,4	2,5	2,8	2,4	2,4	2,6	2,7	2,8	2,9	2,8	2,6	3,2
		σ	1,0	0,9	1,1	1,1	0,9	1,0	1,0	1,1	1,1	1,0	1,0	1,1	0,9	1,0	1,0	1,1	1,0	1,0	0,9	0,7	1,0	0,9	1,0	0,9	0,9	0,9	0,9	0,9	1,0
	Telecommunications	*AA*	3,7	3,7	3,8	3,2	3,6	3,1	3,3	3,2	3,5	3,5	3,5	3,3	2,4	2,0	2,2	2,3	2,1	3,2	2,5	2,8	2,4	2,3	2,4	2,7	3,0	2,8	2,9	2,9	3,2
		σ	0,8	0,9	0,9	1,0	0,7	1,2	1,0	1,3	1,0	1,0	1,0	1,0	1,0	1,0	1,1	1,1	1,1	1,1	1,0	0,8	1,1	1,1	1,2	0,9	1,0	0,9	0,9	0,9	0,9
	Automotive	*AA*	3,7	3,5	3,3	3,6	3,4	3,6	3,4	3,9	3,5	3,4	3,6	3,2	2,8	2,6	2,1	2,6	2,1	3,3	2,4	2,7	2,7	2,8	3,0	2,8	2,6	2,7	2,4	2,4	3,1
		σ	1,0	1,1	1,3	1,3	1,0	1,0	1,0	1,0	1,1	1,2	1,0	1,0	1,0	1,0	1,2	1,2	1,2	1,1	0,9	0,9	1,0	1,1	1,2	1,0	0,8	0,8	0,9	1,1	1,0
	Manufacturing	*AA*	3,8	3,6	3,5	3,2	3,4	3,5	3,3	3,6	3,4	3,4	3,5	3,4	2,9	2,6	2,3	2,8	2,1	3,2	2,4	2,7	2,6	2,8	2,7	2,8	3,0	2,8	2,8	2,6	3,2
		σ	0,9	0,9	1,2	1,3	0,9	1,0	0,9	1,1	0,9	0,9	0,9	0,9	1,1	1,1	1,1	1,1	1,1	1,0	1,0	1,0	1,0	1,0	1,2	1,0	0,8	0,9	1,0	1,0	1,1
	Energy	*AA*	3,9	3,7	3,8	3,4	3,8	3,3	3,3	3,8	3,8	3,6	3,6	3,4	2,7	2,5	2,2	2,7	2,3	3,1	2,4	2,7	2,6	2,6	2,5	2,7	2,8	2,8	2,8	2,6	3,2
		σ	0,9	0,8	1,0	1,1	1,0	1,0	0,8	1,0	0,9	0,8	0,8	0,9	1,1	1,1	0,9	1,0	1,2	0,9	0,9	0,9	1,0	1,0	1,0	0,8	0,9	1,0	1,0	1,0	1,1
	IT Security	*AA*	3,6	3,6	4,0	3,7	3,8	3,0	3,2	3,4	3,5	3,7	3,7	3,3	2,7	2,7	2,3	2,7	2,4	3,3	2,6	2,7	2,7	2,7	2,8	2,9	2,7	2,8	2,8	2,4	3,0
		σ	0,9	1,0	1,0	1,1	0,8	1,3	1,0	1,2	1,0	1,0	1,0	1,1	1,1	1,2	1,1	1,1	1,0	1,0	1,0	1,2	1,1	1,3	0,9	1,0	0,9	1,1	0,9	0,9	
Programming Language	C	*AA*	3,6	3,8	3,6	3,3	3,9	3,4	3,6	3,7	3,8	3,8	3,8	3,2	2,7	2,5	2,2	2,5	2,5	3,0	2,6	2,7	2,8	2,7	2,6	2,8	2,5	2,2	2,7	2,7	3,2
		σ	0,9	1,0	1,2	1,2	0,6	1,0	0,7	0,9	0,8	0,8	0,8	1,1	1,0	1,0	1,0	1,0	1,3	0,9	0,9	0,9	1,0	1,2	1,0	0,9	1,0	0,7	1,1	0,9	0,9
	C++	*AA*	3,7	3,6	3,5	3,1	3,5	3,0	3,2	3,4	3,6	3,5	3,6	3,3	2,6	2,4	2,3	2,6	2,1	3,0	2,6	2,8	2,6	2,6	2,7	2,8	2,9	2,8	2,7	2,3	2,8
		σ	1,0	0,9	1,1	1,2	0,9	1,1	0,9	1,0	0,9	0,8	0,8	1,1	1,2	1,1	1,1	1,0	1,1	1,0	1,0	1,0	1,0	1,2	1,2	0,8	1,0	0,9	1,0	0,8	0,9
	Java	*AA*	3,6	3,6	3,2	3,3	3,6	3,0	3,3	3,4	3,3	3,3	3,4	3,4	2,9	2,7	2,3	2,7	2,6	3,1	2,6	2,8	2,8	2,6	2,9	2,6	2,7	3,0	3,1	2,8	3,2
		σ	1,1	1,1	1,3	1,0	1,0	1,1	1,0	1,1	1,2	1,1	1,0	1,0	1,1	1,0	1,0	1,0	1,1	0,9	1,0	0,9	1,1	1,1	1,1	1,0	0,9	0,9	1,1	1,0	0,8
	Python	*AA*	3,6	3,5	3,6	3,5	3,6	3,3	3,3	3,4	3,4	3,6	3,5	3,4	3,2	3,0	2,7	3,0	2,6	3,2	2,5	2,9	3,0	2,8	2,9	2,9	3,0	3,0	3,1	2,9	3,3
		σ	0,9	1,0	1,2	1,0	1,0	1,1	0,9	1,1	0,9	0,9	0,9	0,8	1,1	1,0	1,0	1,1	1,2	1,1	1,1	1,0	1,1	1,0	1,1	1,3	1,0	0,9	1,0	1,0	1,2
	Other	*AA*	3,7	3,6	3,6	3,2	3,5	2,7	2,9	3,1	3,3	3,1	3,3	3,3	2,6	2,4	2,2	2,4	2,3	2,9	2,4	2,8	2,5	2,3	2,7	2,8	3,0	3,1	3,0	2,8	3,3
		σ	0,8	0,9	1,1	1,2	0,9	1,0	1,0	1,1	1,2	1,1	1,1	0,8	1,1	1,2	1,1	1,1	1,1	1,0	0,9	0,9	1,0	1,0	1,2	0,8	0,9	0,9	0,8	0,9	0,9
Work Experience	less than 3	*AA*	3,6	3,7	3,4	3,2	3,6	2,7	3,0	3,3	3,2	3,3	3,3	3,4	3,1	2,9	2,6	2,9	2,5	3,2	2,6	2,9	2,9	2,7	3,0	3,0	2,9	3,1	3,1	2,8	3,3
		σ	1,0	0,9	1,2	1,1	1,0	1,2	1,0	1,2	1,0	1,0	0,9	0,9	1,1	1,1	1,2	1,1	1,0	1,1	1,0	1,0	1,1	1,1	1,1	1,1	0,9	0,9	1,0	1,0	0,9
	3 to 5	*AA*	3,9	3,4	3,5	3,2	3,4	3,4	3,5	3,3	3,4	3,5	3,8	3,6	3,2	2,8	2,7	2,9	2,6	3,0	2,8	3,1	3,1	3,0	3,0	3,0	3,1	3,2	3,4	3,1	3,3
		σ	0,7	0,9	1,2	1,1	0,8	1,1	0,9	1,1	0,8	0,9	0,9	0,9	1,1	1,0	1,1	1,0	1,2	1,1	1,0	0,9	1,0	1,0	1,1	0,9	0,9	1,0	1,0	1,0	1,0
	6 to 10	*AA*	3,3	3,6	3,3	3,2	3,5	3,3	3,5	3,6	3,5	3,5	3,6	3,1	2,7	2,4	2,2	2,3	2,6	2,9	2,6	2,6	2,6	2,6	2,8	2,7	2,9	2,6	2,8	2,7	3,1
		σ	1,0	1,0	1,2	1,1	0,8	0,8	0,8	0,9	1,0	1,0	0,9	1,0	1,0	0,9	0,8	0,9	1,2	1,0	1,1	0,9	0,9	1,0	1,0	0,9	1,0	0,9	1,0	0,9	1,0
	more than 10	*AA*	3,7	3,7	3,7	3,3	3,8	3,1	3,2	3,4	3,7	3,6	3,6	3,3	2,5	2,4	2,1	2,5	2,2	3,0	2,4	2,7	2,5	2,4	2,5	2,5	2,6	2,5	2,6	2,4	2,9
		σ	0,9	1,0	1,1	1,2	1,0	1,1	1,0	1,1	1,0	1,0	1,0	1,0	1,1	1,2	1,0	1,0	1,2	1,0	0,9	0,9	1,1	1,1	1,2	0,8	0,9	0,8	0,9	0,8	0,9

Maximum in Column Minimum in Column Maximum in Row Minimum in Row

NOTE: AA - **average agreement**, based on Likert scale (1↔SD, 2↔D, 3↔N, 4↔A, 5↔SA), σ - **variance** of average agreement

the different standards are known to a larger percent of general population participants than those in the population that *should know the standard* – this is an issue. It means that the population that should be more aware of these SCG standards is not aware of them. In particular, from the C and C++ software developers, only 28.9% know the SEI-CERT standard, 38.6% know the MISRA standard, and 16.9% know the Annex J of the C11 standard. For developers using Python and Java, 40% know the OWASP standard, and 5.3% know the BSI 5.21 standard. This last result is not surprising, since the BSI standard is local to Germany only, and the survey was deployed on a global scale.

C. Policy Compliance, Neutralization Theory, and Security-Related Stress

Table IV shows the overall results for each theory (policy compliance, neutralization theory, and security-related-stress) for each theory construct, grouped by industry, programming language, and by work experience. The minimum and maximum values are highlighted in this table, with the colors red and green, respectively. For a given group, the minimum and maximum in a column (i.e., per theory construct) is highlighted by a thicker border, while the minimum and maximum in a row is highlighted with a background color (red and green respectively).

In terms of policy compliance, we observe that the highest amount of agreement across all twelve constructs is obtained for the C programming language, while the highest amount of disagreement is obtained for other programming languages and participants with less than three years of industry experience. We attribute the latter observation to the fact that newer employees need to accommodate to the job and might, therefore, not be yet fully integrated into the daily working life. The construct that was rated with the lowest agreement across all

the different groups is PC-NT (i.e. lack of time), This result is to be expected due to the need to fulfill project deadlines in an industrial environment.

In terms of neutralization theory, the construct N-DON3 sees the largest amount of disagreement, i.e., software developers do not think that secure coding guidelines should be ignored to deliver work-packages faster. However, there is a general agreement across all groups (industry, programming languages, and work experience) that ignoring unreasonable secure coding guidelines is acceptable. This result is surprising since, according to the first author's experience, it is not the software developers' job to question the secure coding guidelines but comply with their policies when developing software.

Another surprising factor is that participants in the telecommunications industry find fewer reasons not to comply with secure coding guidelines. According to the first author's experience, this might be because engineers working in this industry are used to developing software under tight constraints (e.g., real-time) and follow established coding guidelines to achieve this goal. However, the IT security and finance department find more reasons not to comply with secure coding guidelines than other industries. This fact is also surprising, especially for the IT security industry. We think that, since the developers working in this industry face security topics daily, they might be more inclined to bend the established rules. Another surprising factor is that, compared to the other programming languages, Python developers tend to find more reasons not to comply with secure coding guidelines. We attribute this observation to the fact that Python is a prototyping language, where software developers might be more used to writing "quick and dirty" code, other than in the other cases. Also, surprisingly, is the fact that software developers using programming languages other than C, C++, Java, or Python find fewer reasons not to comply with secure coding guidelines. In terms of work experience, senior employees (more than ten years experience) tend to follow the established rules, while employees working for three to five years in the industry find more reasons to discard SCG. Another result from this table is that, across all the groups, software developers also tend to ignore SCG that they do not understand.

Finally, in terms of security-related-stress, there is a general agreement on the construct UC4, i.e., the participants to the survey have observed constant changes in security-related technologies. This observation might be related to the large and growing amount of different software development frameworks and changing (agile) software development methodology. However, there is also a general disagreement on UC1, i.e., that secure coding guidelines are not continually changing. We find this last observation positive since constantly changing secure coding guidelines can lead to unnecessary stress at work.

D. Awareness

For each of the programming languages, the participants were asked to answer Yes, Unsure, and No on how they agree with each of the awareness constructs (Per1, Prot1, Be1), and also on BgK5. Additionally, each of these questions was asked in relation to a top-3 CWEs (Common Weakness Enumeration) that affects the programming language, according to the study by WhiteSource [34]. Table V shows the survey results for these constructs, for each programming language and each CWE. We note that each of the CWE is related to one or more secure coding guideline [18]. In this table, an "unsure" answer was considered a negative aspect, therefore combined with "no" results.

The survey participants report high levels of awareness for BgK5 (knowing the vulnerability), and for the construct Prot1 (understanding the consequences of exploiting vulnerabilities). However, for Per1 (ability to recognize vulnerable code) and Be1 (knowing how to write secure code), the levels of awareness are low (less than 51%). The first result is in line with the study by Patel et al. [22], however, the second result is new in this study. For the construct Per1, we also observe that the programming languages "Other", Python and C are especially at risk since the awareness level is low for their ranked vulnerabilities.

Considering all the constructs together, we observe an overestimation (60% vs 40%) of the participants' awareness level, since real-world data shows that the number of incidents is increasing. We attribute this to *optimism bias* [39], which is a well-known effect in risk perception that occurs when someone overestimates or underestimates risk while remaining ignorant about their poor assessment [40]. Our results indicate an overestimation bias, which is corroborated with the industry's experience from the first author.

Since the participants were only asked to rank the top-3 CWE, in BgK4 we asked the participants to optionally name additional weaknesses that they pay attention to while developing software. The participants' answers were coded to separate the correctly identified weaknesses from the vulnerabilities and general issues not related to secure coding. Table VI shows the result of the codification of the answers given by the participants.

From all the additional survey answers, 63.2% are software weaknesses, 33.3% are general coding issues, and 3.5% of the answers are from unsure participants. Python has the highest amount of correctly identified weaknesses and, surprisingly, C++ the least amount. The reason for the last observation might be due to the complexity of the C++ language. In terms of correctly identified issues, Authentication and Authorization, Information leakage, Memory Issues, Weak Cryptography, and Buffer Overflow are in the top-5. Surprisingly, the survey participants have considered general bugs, performance issues, and security breaches as secure coding weaknesses. Denial-of-service is generally not considered a coding issue but a deployment issue (solved with e.g., load balancing). However, it was also considered a software vulnerability, ranking in the top-5 of the general issues category. Surprisingly, also considered as secure coding issues have been: infrastructure issues, safety aspects, and code smells, personal identifiable information, and lack of

TABLE V
AWARENESS RESULTS VS PROGRAMMING LANGUAGE

		Perception (Per1)			W.Avg.	Protection (Prot1)			W.Avg.	Behavior (Be1)			W.Avg.	BgK5			W.Avg.
		Yes	Unsure	No		Yes	Unsure	No		Yes	Unsure	No		Yes	Unsure	No	
C	1) CWE 119	17 (52%)	14/2 (48%)		0,73	30 (91%)	1/2 (9%)		0,92	23 (70%)	7/3 (30%)		0,80	29 (88%)	4/0 (12%)		0,94
	2) CWE 20	22 (67%)	11/0 (33%)		0,83	29 (88%)	4/0 (12%)		0,94	26 (79%)	7/0 (21%)		0,89	30 (91%)	2/1 (9%)		0,94
	3) CWE 399	15 (45%)	13/5 (55%)		0,65	24 (73%)	6/3 (27%)		0,82	16 (48%)	14/3 (52%)		0,70	25 (76%)	6/2 (24%)		0,85
	Average	54 (55%)	38/7 (45%)			83 (84%)	11/5 (16%)			65 (66%)	28/6 (34%)			84 (85%)	12/3 (15%)		
C++	1) CWE 119	29 (58%)	16/5 (42%)		0,74	43 (86%)	3/4 (14%)		0,89	27 (54%)	20/3 (46%)		0,74	44 (88%)	3/3 (12%)		0,91
	2) CWE 20	35 (70%)	11/4 (30%)		0,81	42 (84%)	5/3 (16%)		0,89	34 (68%)	12/4 (32%)		0,80	42 (84%)	5/3 (16%)		0,89
	3) CWE 200	22 (44%)	23/5 (56%)		0,67	34 (68%)	11/5 (32%)		0,79	24 (48%)	21/5 (52%)		0,69	38 (76%)	10/2 (24%)		0,86
	Average	86 (57%)	50/14 (43%)			119 (79%)	19/12 (21%)			85 (57%)	53/12 (43%)			124 (83%)	18/8 (17%)		
Java	1) CWE 20	23 (61%)	12/3 (39%)		0,76	28 (74%)	10/0 (26%)		0,87	21 (55%)	14/3 (45%)		0,74	24 (63%)	13/1 (37%)		0,80
	2) CWE 200	19 (50%)	13/6 (50%)		0,67	28 (74%)	7/3 (26%)		0,83	20 (53%)	12/6 (47%)		0,68	29 (76%)	8/1 (24%)		0,87
	3) CWE 79	15 (39%)	19/4 (61%)		0,64	21 (55%)	13/4 (45%)		0,72	15 (39%)	15/8 (61%)		0,59	24 (63%)	10/4 (37%)		0,76
	Average	57 (50%)	44/13 (50%)			77 (68%)	30/7 (32%)			56 (49%)	41/17 (51%)			77 (68%)	31/6 (32%)		
Python	1) CWE 20	18 (49%)	16/3 (51%)		0,70	26 (70%)	8/3 (30%)		0,81	16 (43%)	16/5 (57%)		0,65	24 (65%)	10/3 (35%)		0,78
	2) CWE 264	20 (54%)	14/3 (46%)		0,73	24 (65%)	10/3 (35%)		0,78	17 (46%)	16/4 (54%)		0,68	12 (32%)	17/8 (68%)		0,55
	3) CWE 79	11 (30%)	19/7 (70%)		0,55	17 (46%)	12/8 (54%)		0,62	12 (32%)	17/8 (68%)		0,55	20 (54%)	12/5 (46%)		0,70
	Average	49 (44%)	49/13 (56%)			67 (60%)	30/14 (40%)			45 (41%)	49/17 (59%)			56 (50%)	39/16 (50%)		
Other	1) CWE 1211	12 (33%)	16/8 (67%)		0,56	19 (53%)	12/5 (47%)		0,69	13 (36%)	16/7 (64%)		0,58	21 (58%)	11/4 (42%)		0,74
	2) CWE 137	14 (39%)	19/3 (61%)		0,65	27 (75%)	6/3 (25%)		0,83	15 (42%)	14/7 (58%)		0,61	28 (78%)	6/2 (22%)		0,86
	3) CWE 200	14 (39%)	17/5 (61%)		0,63	24 (67%)	8/4 (33%)		0,78	11 (31%)	17/8 (69%)		0,54	24 (67%)	9/3 (33%)		0,79
	Average	40 (37%)	52/16 (63%)			70 (65%)	26/12 (35%)			39 (36%)	47/22 (64%)			73 (68%)	26/9 (32%)		
Total Average		286	233	63		416	116	50		290	218	74		414	126	42	
		49%	51%			71%	29%			50%	50%			71%	29%		
Overall Average		1406	693	229													
		60%	40%														

Legend	
	from 55% to 100%
	from 45% to 55%
	from 0% to 45%

W. Avg.	Weighted Average
CWE	Common Weakness Enumeration
Note:	CWEs sorted by incidence ranking, according to WhiteSource study

TABLE VI
BGK4: ADDITIONAL KNOWLEDGE ON CODING WEAKNESSES

		C	C++	Java	Python	Other	Total	Total in (%)
Correctly identified weaknesses		61%	52%	65%	78%	56%		
Weaknesses. (63,2%)	Authentication and authorization Issues	1	2	5	2		10	8,8%
	Information leakage		1	3	3	2	9	7,9%
	Memory issues (e.g. dynamic memory)	3	2	1	2		8	7,0%
	Weak cryptography	1	2	1	5		9	7,9%
	Buffer overflow	1	2		2	1	6	5,3%
	Injection (e.g. SQL)	1		2	2		5	4,4%
	Integer problems	3	2				5	4,4%
	Cross-site request forgery				3	1	4	3,5%
	3rd party components and libraries		1	1		1	3	2,6%
	Logic and exception problems			2	1		3	2,6%
	Input validation		1	1	1		3	2,6%
	Insecure default configuration	1		1		1	3	2,6%
	Data integrity					2	2	1,8%
	Cross-site scripting					2	2	1,8%

		C	C++	Java	Python	Other	Total	Total in (%)
Vulnerabilities and general coding issues		39%	40%	35%	22%	33%		
General Issues (33,3%)	General bugs	3	2	3	2		10	8,8%
	Performance issues (e.g. real-time constraints)	1	3				4	3,5%
	Security breaches	1	1	1		1	4	3,5%
	Denial-of-service		1	1		1	3	2,6%
	Credentials management	1			1		2	1,8%
	Infrastructure issues			1		1	2	1,8%
	Malware		1			1	2	1,8%
	Privilege escalation			1	1		2	1,8%
	Remote or arbitrary code execution		1		1		2	1,8%
	Safety aspects	1		1			2	1,8%
	Insufficient testing				1	1	2	1,8%
	Data loss					1	1	0,9%
	Existence of Personal Identifiable Information		1				1	0,9%
	Code smells			1			1	0,9%

Unsure Answers: **C++** (8%), **Others** (11,1%)

testing. In particular, code smells, which are *symptoms of poor design and implementation choices that may hinder code comprehensibility and maintainability* [41], have been shown to be generally dissociated from security vulnerabilities [42]. The consideration of these factors as secure coding weaknesses leads us to notice the lack of awareness of secure coding guidelines.

E. Actionable Items for Industrial Practitioners

In the following, we present the main actionable items (AI) we infer from the current work, i.e. from the analysis of the survey results but also from our experiences surrounding this topic in industrial application. These actionable items should be taken into consideration by practitioners. We split them into two main categories: general issues and secure coding guidelines. The AIs under the general issues category are not directly related to secure coding guidelines and include:

1) **Need to involve management**: without management understanding and approval, it is not possible to establish secure coding practices in a company
2) **Need to improve knowledge on company's internal S-SDLC and secure coding policies**: the survey has shown that software developers are not always aware of the company's internal policies and about the S-SDLC; therefore, specialized internal campaigns should be started to raise awareness of these issues
3) **Raise awareness of the difference between secure coding and other aspects, e.g. safety and performance**: the survey has shown that software developers tend to

confuse these topics. When training software developers, the difference between these aspects should be made clear, as also possible opposing recommendations

4) **Consider security in the requirements phase**: it is no surprise that security should be considered early in the software development phases; rarely will customers "ask for security"; however, they will expect secure products. Therefore, company policies should be adapted to cover security from early stages, and software developers should be aware of the necessary steps to take (e.g., threat and risk analysis, secure architecture, security requirements).

Furthermore, software developers must be made aware of the difference between security weakness and security vulnerability. In particular, security weaknesses are coding errors that might lead to a vulnerability, while (according to ENISA [43], definition G52), a security vulnerability is the existence of a security weakness that can lead to a security breach. In terms of secure coding guidelines, we conclude the following key AIs for practitioners:

1) **Include SCG as an integral part of S-SDLC**: SCG should be lived as a process and should be second nature to software developers; daily practice and usage has the potential to have long-lasting and beneficial effects.
2) **Build a secure coding community**: promote secure coding practices inside the company. Some possible ways to implement this it to join larger communities which also promote secure coding practices, e.g., SAFEcode [44], have monthly or weekly presentations or discussions on a secure coding topic, promote and use secure coding gamification (e.g., best secure coder of the month).
3) **Define a responsible person**: have a point-of-contact for secure coding issues; the job of this person includes making sure that software developers are trained, and motivated.
4) **Implement awareness training on SCG**: a substantial amount of software developers needs training on SCG; as such, awareness training events should be promoted and held regularly.
5) **Implement hands-on awareness training**: motivated by the observed optimism bias, and also experience from the industry, we think that an effective way to raise awareness is to challenge the knowledge of software developers on secure coding topics; while desired training methods was not captured through the survey, our experience has shown that after being challenged on these topics, many developers tend to re-evaluate their knowledge and seek more information. A good way to achieve this is by the usage of Capture-the-Flag events, which are specially designed to raise awareness of secure coding for software developers in the industry; these exercises should mainly focus on the defensive perspective but also cover offensive aspects; furthermore, these exercises should provide a good motivation on why certain SCG exist – this way, software developers can develop a better knowledge of SCG and understanding on why they should comply to them.
6) **Implement SCG quality gates**: secure code is also high-quality code; practitioners should consider adding the requirement of checking secure coding guidelines to typical project quality gates; some ways this can be achieved include using specialized tools that are configured to check secure coding guidelines, keeping track of code reviews including review of secure coding guidelines, and status monitoring of software security testing results.
7) **Do not use SAST as a replacement for SCG**: some participants to the survey have mentioned the usage of SAST tools as a replacement for a formal training or consideration of secure coding guidelines; we consider this an important hidden danger – previous studies [45], [46] have reported on the poor quality of SAST tools; therefore, we conclude that the human factor cannot be taken out of the loop and SAST tools should only be used in a supportive role.
8) **Monitor the quality of SAST tools**: since the quality of the output of these tools might be poor, strategies to address their quality needs to be considered; in particular, the secure coding champion needs to implement a process to verify the quality of the tools being deployed and to replace them when outdated; additionally, if possible, it should be considered to use several tools in parallel, in order to compare the different results.
9) **Training on SCG should focus on concepts, not specific cases or instances**: the results of the survey indicate constant changes in security technologies; this might be related to the vast amount of existing frameworks and booming IT security field; these changes can cause unnecessary stress to software developers; as such, when dealing with secure coding guidelines, software developers should be trained on concepts and not so much on particular instances of SCG, in particular, software developers should understand the underlying reason for the SCG and not just assume the rule without further consideration.
10) **Keep up-to-date with the latest technology**: software developers should be informed about the latest security technologies, when necessary, especially when starting new projects; this should be done taking into consideration that too much information can cause stress, while too little information can mean that important news are missed; for this reason, we propose that the secure coding champion should constantly monitor new technologies and decide on their importance and introduction on running projects.
11) **Adapt SCG, only when necessary**: similar to keeping up-to-date with the latest technology, secure coding guidelines should be updated regularly; however, without interfering with ongoing projects; updates of SCG should be timed together with other SCG awareness campaigns, e.g., awareness training.

F. Threats to Validity

In this work, we present the analysis of a large scale anonymous survey on the usage of secure coding guidelines in the industry, including a total number of 194 participants distributed across the globe. Since the survey took place in an online format, and the collected data is anonymized, it is impossible to control the respondents' true background. However, we have counter-acted this possible bias in two different ways: by making sure that the channels where the survey was announced included a rich set of industrial software developers and that this requirement was clearly stated at the start of the survey (in particular with the following sentence in the beginning of the survey: "this survey is for software developers working in the industry"). Geographical background was not captured, which might impact our conclusions. Additionally, the different industry sectors are not equally represented, which might introduce bias to our conclusions. Our conclusions result from an interpretation of the data in light of the first author's own experience in the industry. However, these results have been discussed with three additional security experts, whereby the conclusions hereby presented have been confirmed by all. We observe that the survey results display optimism bias. To counterbalance this effect, we focus our results, not on absolute values, but on a relative comparison between different values. Finally, the results on knowledge of the BSI standard might have a strong bias, since this is a local standard to Germany, and geographic results are not available.

G. Impact of this work

The results presented in this work provide an impact both in the academic community but also in the industry. It is not always easy to obtain a large volume of survey data from participants from the industry. In this work, we have collected the survey answers from over 190 industrial software developers. This means that we can assume that our results have a strong supportive basis. Also, the survey that was administered to the participants underwent an extensive design cycle, to make sure that it is based on well established scientific theories. As a result of the analysis of the survey data, fifteen key take-away messages were derived, which serve both as guidance for future scientific work but also as valuable information for industry practitioners. Our work not only addresses the awareness of software developers on the topic of secure coding guidelines, but it also raises awareness in the scientific community on this difficult topic. Additionally, we provide the raw survey data, as a means to contribute to further research on the topic.

V. Conclusions

Cybersecurity is becoming ever more important nowadays. Ignoring cybersecurity can lead to severe financial penalties or even loss of certification, together with loss of business or even loss of life, in critical infrastructures. However, the last years have seen an increase in cybersecurity incidents. According to an estimate by the United States Department of Homeland Security, the root cause of about 90% of security incidents can be traced back to software design and coding weaknesses. Secure coding guidelines exist to make software secure – compliance to them increases the security and quality of code. Additionally, Static Application Security Test tools also exist to reduce software vulnerabilities; however, previous studies have shown that these tools exhibit many false positives and false negatives. These facts lead us to the following questions: 1) how aware are software developers of secure coding guidelines, 2) is secure coding education in the industry needed, and which factors lead software developers to comply or ignore secure coding guidelines.

In this work, we address these questions through a large-scale survey on software developers in the industry. The survey design, which is a complicated endeavor by itself, is addressed in a separate publication, while this focuses on the analysis of the results and practical aspects and advice for practitioners and cybersecurity educators in the industry. Our measurement of policy compliance is based on three established theories: policy compliance theory by Bulgurcu et al. and Moody et al.; neutralization theory by Siponen et al.; and security-related stress theory by D'Arcy et al. Our measurement of awareness is based on the three dimensions, as defined by Hänsch et al.: perception, protection, and behavior.

Our results indicate that a large amount of software developers are not aware of secure coding guidelines. Previous studies argue that increasing awareness leads to increased compliance. Therefore, we conclude that a method to address this lack of awareness is through education on secure coding. Based on our results and experience, we also infer a set of fifteen actionable items for practitioners and industrial cybersecurity educators. A further contribution herein is the raw survey results, which we make openly available for further research. In future work, we would like to practice the derived actionable items and investigate novel methodologies for secure coding education of industrial software developers. We want to use these actionable items to improve our ongoing action-design research in the industry.

Supporting Data

The raw data collected in the survey is openly available in Zenodo [38]. The raw survey data is provided in Comma Separated Values (CSV) format. Researchers are encouraged to make use of this data for further work.

Acknowledgements

The authors would like to thank the participants of the survey for their constribution. Also, the authors would like to thank Kristian Beckers and Thomas Diefenbach for their helpful, insightful, and constructive comments and discussions.

This work is financed by portuguese national funds through FCT - Fundação para a Ciência e Tecnologia, I.P., under the project FCT UIDB/04466/2020. Furthermore, the third author thanks the Instituto Universitário de Lisboa and ISTAR-IUL, for their support.

References

[1] Kaspersky, "The State of Industrial Cybersecurity – 2017," 2017. [Online]. Available: https://tinyurl.com/y7dfppak

[2] Department of Homeland Security, US-CERT, "Software Assurance," Sep. 2020. [Online]. Available: https://tinyurl.com/y6pr9v42

[3] ISO 27001, "Information technology – Security techniques – Information security management systems – Requirements," International Standard Organization, Geneva, CH, Standard, Oct. 2013.

[4] IEC 62443-4-1, "Security for industrial automation and control systems - part 4-1: Secure product development lifecycle requirements," International Electrotechnical Commission, Standard, Jan 2018.

[5] ISO, "ISO 250xx Series," International Organization for Standardization, Geneva, CH, Standard, 2005. [Online]. Available: http://iso25000.com/index.php/en/iso-25000-standards

[6] Carnegie Mellon University, "SEI-CERT Coding Standards." [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/seccode

[7] –, "Guidelines for the use of the C language in critical systems," Motor Industry Software Reliability Association, Nuneaton, Warwickshire, UK, Standard, Mar 2012.

[8] ——, "Additional security guidelines for MISRA C:2012," Motor Industry Software Reliability Association, Nuneaton, Warwickshire, UK, Standard, Mar 2016.

[9] "OWASP Top 10," Jul. 2017. [Online]. Available: https://tinyurl.com/yyb8wcv9

[10] SAFECode Charter Members, "SAFECode - Software Assurance Forum for Excellence in Code," accessed Mar. 2020. [Online]. Available: https://safecode.org

[11] M. Rodriguez, M. Piattini, and C. Ebert, "Software verification and validation technologies and tools," *IEEE Software*, vol. 36, no. 2, pp. 13–24, 2019.

[12] T. D. Oyetoyan, B. Milosheska, M. Grini, and D. S. Cruzes, "Myths and facts about static application security testing tools: an action research at Telenor digital," in *International Conference on Agile Software Development*. Springer, Cham, 2018, pp. 86–103.

[13] T. Gasiba, K. Beckers, S. Suppan, and F. Rezabek, "On the Requirements for Serious Games geared towards Software Developers in the Industry," in *Conference on Requirements Engineering Conference*, D. E. Damian, A. Perini, and S. Lee, Eds. Jeju, South Korea: IEEE, Sep. 2019, pp. 286–296. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome/8910334/proceeding

[14] T. Gasiba, U. Lechner, M. Pinto-Albuquerque, and D. M. Fernandez, "Awareness of Secure Coding Guidelines in the Industry - A first data analysis," in *TrustCom 2020: International Conference on Trust, Security and Privacy in Computing and Communications*. Guangzhou, China: IEEE, Dec. 2020.

[15] T. Gasiba, U. Lechner, M. Pinto-Albuquerque, and A. Zouitni, "Design of Secure Coding Challenges for Cybersecurity Education in the Industry," *13th International Conference on the Quality of Information and Communications Technology, QUATIC*, pp. 223–237, 09 2020.

[16] T. Gasiba, U. Lechner, M. Pinto-Albuquerque, and A. Porwal, "Cybersecurity Awareness Platform with Virtual Coach and Automated Challenge Assessment," in *6th Workshop On The Security Of Industrial Control Systems & Of Cyber-Physical Systems (CyberICPS)*. Online: Springer, Cham, 12 2020, pp. 67–83.

[17] T. Gasiba, U. Lechner, and M. Pinto-Albuquerque, "Sifu - A CyberSecurity Awareness Platform with Challenge Assessment and Intelligent Coach," in *Cybersecurity Journal, Special Issue on Cyber-Physical System Security*. SpringerOpen, 12 2020, pp. 1–23.

[18] T. Gasiba, U. Lechner, J. Cuellar, and A. Zouitni, "Ranking Secure Coding Guidelines for Software Developer Awareness Training in the Industry," in *First International Computer Programming Education Conference (ICPEC 2020)*, ser. OpenAccess Series in Informatics (OASIcs), R. Queirós, F. Portela, M. Pinto, and A. Simões, Eds., vol. 81. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 11:1–11:11.

[19] T. Gasiba and U. Lechner, "Raising Secure Coding Awareness for Software Developers in the Industry," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. Jeju, South Korea: IEEE, Sep. 2019, pp. 141–143.

[20] T. Gasiba, U. Lechner, F. Rezabek, and M. Pinto-Albuquerque, "Cybersecurity Games for Secure Programming Education in the Industry: Gameplay Analysis," in *First International Computer Programming Education Conference (ICPEC 2020)*, ser. OpenAccess Series in Informatics (OASIcs), R. Queirós, F. Portela, M. Pinto, and A. Simões, Eds., vol. 81. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 10:1–10:11.

[21] T. Gasiba, U. Lechner, and M. Pinto-Albuquerque, "CyberSecurity Challenges: Serious Games for Awareness Training in Industrial Environments," 2 2021, in Bundesamt für Sicherheit in der Informationstechnik (Hg.): Deutschland. Digital. Sicher. 30 Jahre BSI – Tagungsband zum 17. Deutschen IT-Sicherheitskongress.

[22] S. Patel, "2019 Global Developer Report: DevSecOps finds security roadblocks divide teams," Jul. 2020. [Online]. Available: https://tinyurl.com/y6oypsh3

[23] B. Schneier, "Software Developers and Security," Online, Jul. 2020, https://www.schneier.com/blog/archives/2019/07/software_develo.html.

[24] Sourcegraph, "The Emergence of Big Code – A 2020 Survey of Software Professionals," Oct 2020. [Online]. Available: https://tinyurl.com/y5yfprn8

[25] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE. an Jose, CA: IEEE, 2017, pp. 121–136.

[26] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers need support, too: A survey of security advice for software developers," in *2017 IEEE Cybersecurity Development (SecDev)*. Cambridge, MA, USA: IEEE, Sep. 2017, pp. 22–26.

[27] H. Assal and S. Chiasson, "'Think secure from the beginning' A Survey with Software Developers," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–13.

[28] J. Xie, H. R. Lipford, and B. Chu, "Why do Programmers Make Security Errors?" *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 161–164, Sep. 2011.

[29] B. Bulgurcu, H. Cavusoglu, and I. Benbasat, "Information Security Policy Compliance: An Empirical Study of Rationality-Based Beliefs and Information Security Awareness," *MIS quarterly*, vol. 34, no. 3, pp. 523–548, 2010.

[30] G. D. Moody, M. Siponen, and S. Pahnila, "Toward a Unified Model of Information Security Policy Compliance," *MIS quarterly*, vol. 42, no. 1, pp. 1–50, 2018.

[31] M. Siponen and A. Vance, "Neutralization: New Insights into the Problem of Employee Information Systems Security Policy Violations," *MIS quarterly*, vol. 34, no. 3, pp. 487–502, 2010.

[32] J. D'Arcy, T. Herath, and M. K. Shoss, "Understanding Employee Responses to Stressful Information Security Requirements: A Coping Perspective," *Journal of management information systems*, vol. 31, no. 2, pp. 285–318, 2014.

[33] N. Haensch and Z. Benenson, "Specifying IT security awareness," in *25th International Workshop on Database and Expert Systems Applications, Munich, Germany*. Munich, Germany: IEEE, Sep 2014, pp. 326–330.

[34] WhiteSource, "What are the Most Secure Programming Languages?" Mar. 2019, https://tinyurl.com/y2rmfhn7.

[35] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un)happy," *Journal of Systems and Software*, vol. 140, pp. 32–47, 2017.

[36] MITRE-Corporation, "Common weaknesses enumeration," 2019. [Online]. Available: https://cwe.mitre.org/

[37] C. Schmitz, "LimeSurvey v3.17.0," Apr. 2020. [Online]. Available: https://www.limesurvey.org

[38] *Raw Results for the Preliminary Survey on Awareness of Secure Coding Guidelines in the Industry*. Zenodo, Oct. 2020. [Online]. Available: https://zenodo.org/record/4075282

[39] R. Thaler and C. Sunstein, *Nudge: Improving Decisions About Health, Wealth, and Happiness*. Yale University Press, 2008.

[40] U. Lechner, "IT-Security in Critical Infrastructures Experiences, Results and Research Directions," in *International Conference on Distributed Computing and Internet Technology*. Springer, 2019, pp. 42–59.

[41] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, "On the Diffuseness and the Impact on Maintainability of Code Smells: A Large Scale Empirical Investigation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1188–1221, 2018.

[42] A. A. Elkhail and T. Cerny, "On Relating Code Smells to Security Vulnerabilities," in *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance*

and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS). IEEE, 2019, pp. 7–12.

[43] European Union Agency for Cybersecurity (ENISA), "Risk Management Glossary," Oct. 2020. [Online]. Available: https://tinyurl.com/y329vqmb

[44] Software Assurance Forum for Excellence in Code, "SAFECode - Fundamental Practices for Secure Software Development - Essential Elements of a Secure Development Life-cycle Program, 3rd Ed." 03 2018. [Online]. Available: https://tinyurl.com/y44etrs7

[45] B. Aloraini, M. Nagappan, D. M. German, S. Hayashi, and Y. Higo, "An Empirical Study of Security Warnings From Static Application Security Testing Tools," *Journal of Systems and Software*, 2019.

[46] J. Li, "Vulnerabilities Mapping based on OWASP-SANS: A Survey for Static Application Security Testing (SAST)," *Annals of Emerging Technologies in Computing*, 2020.