Ranking Secure Coding Guidelines for Software Developer Awareness Training in the Industry

Tiago Gasiba
Siemens AG, München, Germany
Universität der Bundeswehr München, Germany
tiago.gasiba@siemens.com

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

Jorge Cuellar
Siemens AG, München, Germany
Universität Passau, Germany
jorge.cuellar@siemens.com

Alae Zouitni
Universität Passau, Germany
zouitni.alae@gmail.com

Abstract

Secure coding guidelines are essential material used to train and raise awareness of software developers on the topic of secure software development. In industrial environments, since developer time is costly, and training and education is part of non-productive hours, it is important to address and stress the most important topics first. In this work, we devise a method, based on publicly available real-world vulnerability databases and secure coding guideline databases, to rank important secure coding guidelines based on defined industry-relevant metrics. The goal is to define priorities for a teaching curriculum on raising cybersecurity awareness of software developers on secure coding guidelines. Furthermore, we do a small comparison study by asking computer science students from university on how they rank the importance of secure coding guidelines and compare the outcome to our results.

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

Keywords and phrases education, teaching, training, secure coding, industry, cybersecurity, capture-the-flag, game analysis, game design, cybersecurity challenge

Digital Object Identifier 10.4230/OASIcs.ICPEC.2020.11

Acknowledgements We would like to thank the anonymous reviewers for the valuable comments and careful reviews. We would also like to thank all survey participants as well as our colleagues Holger Dreger and Thomas Diefenbach for many fruitful discussions.

1 Introduction

It is widely known that developers (humans) make mistakes during software development which result in bugs [7, 21, 24, 27, 34]. In particular, these bugs can lead to software vulnerabilities that can result in potentially fatal consequences, both for the user of the software, the owner or service provider and the company that sells the software.

Many security standards, e.g. [8, 5, 6, 26, 29, 25], nowadays mandate the implementation of a secure software development life-cycle (e.g [17]), which aims at significantly reducing the number of vulnerabilities in software. In order to be effective, companies should make sure

that their software developers are properly trained in writing secure software, i.e. secure code. Not only is this an important factor in reducing the number of software vulnerabilities, but this is typically checked during audits and certification. As such, specialized training that addresses how to write secure code is needed in order to raise the cybersecurity awareness [13] of software developers in the industry.

Since training of software developers in the industry costs precious time and money, it makes more sense to focus first the effort of training and raising awareness on issues that cause a larger impact to the business [10]. In particular, we are interested in ranking secure coding guidelines for teaching purposes.

In this work we intend to focus on C and C++ programming languages, since they are currently widely used in the industry [28]. For these two programming languages, we use the well known secure coding guidelines (SCG) from Carnegie Mellon University [31] as the basis for the teaching curriculum.

One possible way to prioritize this curriculum, is to base the ranking of the SCG on two steps: 1) the impact rating from real-world software vulnerabilities and 2) the mapping of vulnerabilities to secure coding guidelines. One common and widely used way to rate the impact of software vulnerabilities is to use the common vulnerability scoring system (CVSS) [15]. Online databases, such as the Common Vulnerabilities and Exposure database [23], give extensive lists of known vulnerabilities (CWE - Common Weakness Enumeration), their CVSS scoring and also provides a mapping from these vulnerabilities to secure coding guidelines.

Towards the goal of prioritizing SCG, we need to compute ranking metrics for secure coding guidelines. Note, however, that the goal of this work is not to establish new metrics for secure coding guidelines, but to understand what are the most important SCG that should be taught during awareness training for industrial software developers. However, due to the lack of well-known metrics that combine all the previously mentioned factors, we first define four different CWE metrics, based on well-known mathematical functions (e.g. average value, weighted average, etc.). These metrics are also based on [23], CVSS scoring and also on discussions with cybersecurity experts.

In order to determine and motivate the need for education of secure coding guidelines, we also try to understand the gap between academia and industry. The main question we would like to address here is: are future industry software developers aware which are the secure coding guidelines that have the most relevancy for the industry (i.e. based on real-world data)? This is done by asking last-year university students of computer-science course (not specific to cybersecurity) to rank secure coding guidelines using a likert scale [18] through the means of a questionnaire.

Our main contributions in this work are the following:

- Methodology to compute ranks of secure coding guidelines as basis for prioritization of the education of software developers
- Tables with ranked secure coding guidelines for C and C++ based on real-world data
- Analysis of different ranked SCG, leading to the conclusion that the exact CVSS score values do not significantly contribute to the ranking
- Comparison study between real-world data and student perception of ranking of secure coding guidelines

2 Related Work

In industry several IT security standards, e.g. [8, 5, 6, 26, 29, 25] mandate not only the implementation of secure coding guidelines into the software development life-cycle but also that the developers are properly trained in secure software development. C and C++ are typical and widely used programming languages in the industry [28]. The major secure coding standards in existence for C and C++ are from the Carnegie Mellon University [31] and from MISRA [2, 3].

The importance of secure programming guidelines in the software development life-cycle is discussed by Tabassum et al. and Whitney et al. in [30] and [32] respectively. In order to to raise awareness [13] of software developers on the topic of secure coding guidelines, they study two different methods: ESIDE, an educational IDE plugin, and coaching by security experts. Their preliminary results give indicators that both methods are suitable towards this goal. In a related work, Gadient et al. [9] develop an IDE plugin to detect security code smells as a supportive measure for (Java) software developers. They found out that, out of the 100 applications they investigated, 44 contained security vulnerabilities.

Additionally, many developers lacking knowledge or training in secure coding tend to search online forums on solutions to secure coding problems [33, 24, 20]. It has been shown that the information provided in these online forums can lead to the introduction of further vulnerabilities into the source code [7, 34], if the developers are not aware on how to write secure code. Furthermore, Meng et al. [21] show that the a substantial number of developers does not appear to understand the security implications of coding options and also links this to the lack of cybersecurity training.

Bagnara et al. [4] discuss the MISRA-C guidelines, which are C-specific guidelines composed of safety guidelines and secure coding guidelines. In their work, they distinguish between guidelines that can be verified by automatic tools such as static code analysis, those that require information that is beyond the reach contained in the source code and those that relate to compliance. In [11], Goodall et al propose a method, based on static code analysis, which can be used by software developers to visualize code security. However, in [12] Goseva et al. point out that the coverage of static code analysis tools can vary across different tools. They conclude that one should not rely only on static code analysis tools, otherwise a large number of vulnerabilities can be left undiscovered.

This further points out the need to train software developers in secure coding guidelines, specifically on high impact rules. If software developers are not aware of secure coding guidelines and the issues that can be caused by exploiting vulnerable code, the effectiveness of such kind of tools will be limited. Results presented by Rexxa et al [27] corroborate with these observations. Although focused on web technologies, their results point out that one major reason for software vulnerabilities is the lack of experience and lack of knowledge about secure coding and secure application development.

Recent research results explore promising, new and innovative ways to raise awareness about secure coding to software developers using capture-the-flag methodology [10, 16]. The results presented in this work are directly applicable to this education methodology as a means to prioritize on developed challenges and awarded game points.

3 Outlook of the work

In this section we give a brief overview of our work. It was done in two phases: Phase 1: ranking of SCG using online databases and Phase 2: raking of SCG through questionnaires administered to academia students. Note, this process was repeated for the C programming language and for the C++ programming language.

3.1 Phase 1: ranking of SCG using online databases

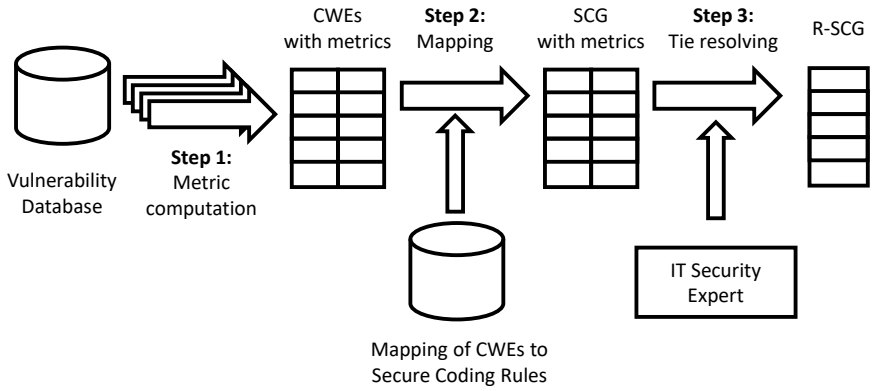

Figure 1 Derivation of Ranked Secure Coding Guidelines.

Figure 1 shows the process that followed in order to derive ranked secure coding guidelines (R-SCG). It consists of the following three steps:

Step 1 compute CWE metric $m^{(x)}(c)$ for each of the four defined metrics (see section 3.3)

Step 2 compute SCG metric based on CWE $\rightarrow$ SCG mapping and then filtering the top 15 SCG by computed metric

Step 3 generate R-SCG table by resolving ambiguous ranks (i.e. using expert opinion for SCG that have the same metric value)

3.1.1 Details on step 1: computing CWE metric

Based on the CVE details online database [23], we have extracted and grouped the CWEs and their corresponding CVSS scores $s(c, \lambda)$. Here c represents the CWE ID and λ represents the observation index, which ranges from 1 to $n(c)$, i.e. the total existing entries (observations) in the database that have CWE with ID= c. At the time that we consulted the online database (May 2019), it consisted of 114.686 observations from 1st January 1999 until 5th May 2019 containing 112 unique CWE identifiers. The computation of the four metric functions $m^{(1)}(c)$ to $m^{(4)}(c)$ will be detailed in section 3.3.

3.1.2 Details on step 2: compute SCG metric based on CWE metric

The MITRE CWE database [22], contains pointers from CWE to the affected SCG from Carnegie Mellon CERT-SEI Secure Coding Guideline database [31]. The mapping provided by this database was used as the mapping rule. Note that, in this database, one CWE can map more than one SCG (see Figure 2). The final SCG metric was taken as the sum of the related CWE metrics multiplied by the CERT-SEI priority level (see sub-section 3.4).

3.1.3 Details on step 3: generate R-SCG table

After step 2, some SCG still had the same computed metric. At this stage, it was decided to disambiguate the tied SCG by gathering input from three different IT cybersecurity experts from the industry. The experts were asked to rank the relative importance of only those guidelines that had the same metric. After this, a table containing the ordered SCG was produced, which we call the ranked secure coding guidelines (R-SCG).

3.2 Phase 2: ranking of SCG by academia students

In order to understand how students in the academia perceive the importance of secure coding guidelines, we have conducted an online questionnaire using Google forms. The number of participants in this questionnaire, which lasted one month and was done in *September 2019*, was 34. The age of the participants ranged from 23 to 30, they were all Master students in computer science (not specializing in cybersecurity), in their second year (last year). All of the participants were familiar with programming in C, and half of them (17 participants) were familiar with programming in C++.

Participants were asked to rank every secure coding rule, which was the outcome of phase 1, in a five point likert scale [18] ranging from "not important" to "very important". For every SCG, the individual likert points were averaged. The resulting SCG were sorted based on the average likert points, resulting in a ranked secure coding guidelines P_C and P_{C++} from academia.

3.3 Metrics

Four different metrics as defined below were used to rate the importance of the CWE in relation to each other (see Table 1). In this table, c represents a CWE ID, $n(c)$ the number of occurrences of incidents in the online database related to c and $s(c, \lambda)$ represents the CVSS score (with values in the range 0..10) present in the online database where c is the attached CWE ID and λ a running index of the entries (in the range $1..n(c)$ entries). A CVSS score is a quantitative severity ranking measure, with 0 being the lowest and 10 being the highest. The reason why four metrics were used, was due to the lack of previous work that gives a metric on SCG based on CVSS scores. Note that all formulas use standard well-know formulas adjusted by the number of occurrences $n(c)$, in order to penalize CWEs that occur more often.

In our work, we define these four metrics as a mean to aggregate the individual CVSS scores into a high-level individual CWE score, i.e. $m^{(x)}(c)$, with x being the selected metric according to Table 1. This metric is then used, as shown in the next sections, to make a further breakdown to individual secure coding guidelines, as shown in Figure 1, step 1.

Table 1 CWE Metrics.

Metric	Description	Formula
#1	Average CVSS Scoring	$m^{(1)}(c) = n(c) \times \frac{\sum_{\lambda=1}^{n(c)} s(c,\lambda)}{n(c)}$
#2	Weighted average CVSS Scoring	$m^{(2)}(c) = n(c) \times \frac{\sum_{\lambda=1}^{n(c)} s^2(c,\lambda)}{\sum_{\lambda}^{n(c)} s(c,\lambda)}$
#3	Worst-case Score	$m^{(3)}(c) = n(c) \times \max_\lambda s(c,\lambda)$
#4	Number of occurrences	$m^{(4)}(c) = n(c)$

3.4 Mapping CWE metrics to SCG metrics

The CWE metrics, as obtained above, are mapped to SCG metrics, as shown in step 2 of Figure 1. In order to achieve this, we used the existing mapping from CWE IDs to SCGs as given by MITRE [22]. It was observed that using this mapping, a single CWE ID can relate to several SCGs. Therefore, we aggregate the final metric computation as given in the exemplary Figure 2. In this example, SCG_1 is referenced by two CWE IDs: CWE_1 and

CWE_3, where each CWE has its own attached metric, as given section 3.3. The resulting SCG metric is then given by $p(SCG_1) \times (m^{(x)}(CWE_1) + m^{(x)}(CWE_3))$, where x in the range $[1, 4]$, and $p(SCG_1)$ represents the SCG priority given by Carnegie Mellon SCG in [31].

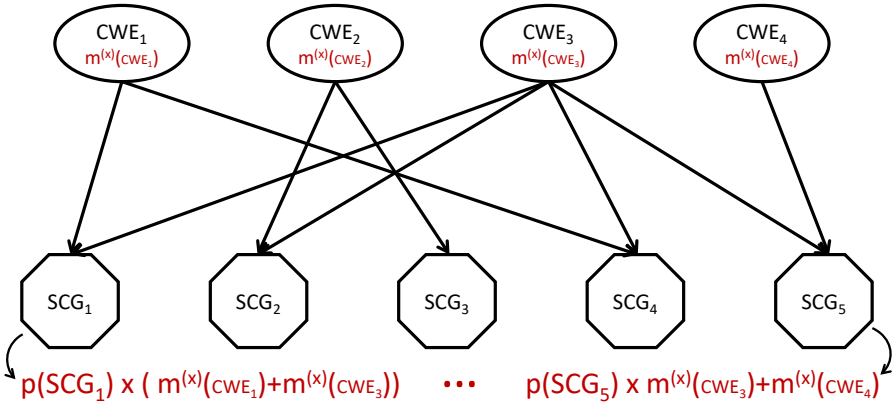

Figure 2 Details on Mapping CWE metrics to SCG metrics.

4 Results

After step 2 and step 3 of phase 1, the results for the C and C++ ranked secure coding guidelines, can be seen in Table 2 and Table 3. Note that in this section, we present the final results, corresponding to step 3 in Figure 1, which are the ranks of the SCG (e.g $R_C^{(1)}$, $R_C^{(2)}$, $R_C^{(3)}$,$R_C^{(4)}$), after computing the different metrics 1..4 for each secure coding guideline. We also present the SCG ranked by the students (P_C) by means of the survey. In the tables, lower numbers indicate higher ranks and higher numbers indicate lower ranks.

Table 2 Top 16 C Ranked Secure Coding Guidelines.

C SCG	$R_C^{(1)}$	$R_C^{(2)}$	$R_C^{(3)}$	$R_C^{(4)}$	P_C
STR38	1	1	1	1	5
EXP34	4	2	2	2	11
STR31	2	3	3	3	2
ARR38	3	4	4	4	6
EXP33	9	5	5	6	12
FIO30	7	6	7	5	1
STR32	8	7	6	7	3
ARR30	12	8	8	10	4
FIO34	10	9	9	8	13
FIO37	11	10	10	9	15
ARR32	13	11	11	11	10
ARR39	14	12	12	12	9
FIO45	16	13	13	13	14
MEM30	5	14	14	14	8
MEM34	6	15	15	15	16
MEM35	15	16	16	16	7

Table 3 Top 15 C++ Ranked Secure Coding Guidelines.

C++SCG	$R_{C++}^{(1)}$	$R_{C++}^{(2)}$	$R_{C++}^{(3)}$	$R_{C++}^{(4)}$	P_{C++}
MEM50	1	1	1	1	5
MEM51	2	2	2	2	7
MEM52	3	3	3	3	1
MEM53	4	4	4	4	3
MEM54	5	5	5	5	4
MEM55	6	6	6	6	15
MEM56	7	7	7	7	10
STR50	8	13	13	13	2
STR51	9	14	14	14	9
EXP53	10	8	8	8	12
EXP60	11	9	9	9	14
EXP54	12	10	10	10	6
EXP61	13	11	11	11	11
EXP62	14	12	12	12	8
STR52	15	15	15	15	13

In order to compare the rankings between themselves, we have computed the Kendall's tau distance metric [19] between the different ranked lists. The Kendall's tau distance is equal to number of exchanges that a bubble sort algorithm needs to apply to one list so that it becomes equal to the other list, i.e. it results in the same ordering of items. A Kendall tau distance of 0 means that the lists contain the elements in the same order. For two lists of size N that are not in the same order, the Kendall tau distance is a value larger than zero and smaller or equal to $N \times (N-1)/2$, i.e. the maximum number of exchanges that a bubble sort algorithm can perform.

The normalized Kendall tau distance values, i.e. the Kendall-tau distance divided by $N \times (N-1)/2$ (possible values ranging from [0..1.0]), is shown in Table 4 and Table 5.

Table 4 Normalized Kendall's tau distance for C SCG.

	$R_C^{(1)}$	$R_C^{(2)}$	$R_C^{(3)}$	$R_C^{(4)}$	P_C
$R_C^{(1)}$	0.000				
$R_C^{(2)}$	0.208	0.000			
$R_C^{(3)}$	0.217	0.008	0.000		
$R_C^{(4)}$	0.183	0.025	0.033	0.000	
P_C	0.379	0.358	0.367	0.367	0.000

Table 5 Normalized Kendall's tau distance for C++ SCG.

	$R_{C++}^{(1)}$	$R_{C++}^{(2)}$	$R_{C++}^{(3)}$	$R_{C++}^{(4)}$	P_{C++}
$R_{C++}^{(1)}$	0.000				
$R_{C++}^{(2)}$	0.095	0.000			
$R_{C++}^{(3)}$	0.095	0.000	0.000		
$R_{C++}^{(4)}$	0.095	0.000	0.000	0.000	
P_{C++}	0.300	0.367	0.367	0.367	0.000

5 Discussion

5.1 Secure Coding Guidelines for C

Table 2 shows the results of the ranked secure coding guidelines for metrics 1..4 and for the students, all for the C programming language. In this table, lower numbers mean higher ranks and larger numbers mean lower ranks. For example, based on Metric 1, the top-5 ranked SCG is [STR38, STR31, ARR38, EXP34, MEM30], while using Metric 2 they are [STR38, EXP34, STR31, ARR38, EXP33]. Addionally, Table 4 shows the corresponding Kendall distance between the R-SCG. For example, the distance between the R-SCG using Metric 1 and Metric 3 is 0.217.

In Table 2 we can see that we can group the obtained results into three different clusters: 1:$\{R_C^{(1)}\}$, 2:$\{R_C^{(2)}, R_C^{(3)}, R_C^{(4)}\}$ and 3:$\{P_C\}$ according to their relative distances. The 3rd cluster is the one that is most distant from all the other clusters, with a distance bigger in the range $]0.35, 0.38[$. Since this cluster represents the feedback given by students, it also means that their answers are the most farther away from our outcome using real-world data. Furthermore, the 1st cluster (Metric 1) is also distant from the 2nd cluster (Metric 2, 3 and 4), whereby the normalized Kendall-tau distance is bigger than $]0.22, 0.25[$. It is surprising that the Metric 2, 3 and 4 have low distance and form a separate cluster to Metric 1. This discrepancy is most likely due to the the fact that the first metric, since it takes the average CVSS score, tends to lower the overall metric value, while all the other metrics penalize on higher CVSS scores, potentially leading to a different sorting of the list. It is nonetheless interesting to note that, for the defined four metrics, the list of the top-4 most important SCG still contain the same guidelines.

The secure coding guidelines which has gotten the highest ranking among the students was FIO30-C, which is "exclude user input from format strings". The same guideline is ranked lower using Metric 1, 2, 3 and 4, having ranks 7, 6, 7, 5 respectively. Although

not following this SCG can obviously lead to vulnerabilities, in order to exploit it, several additional conditions must be met - this is reflected, in practice, by the lower rank achieved by the results based on real-world data.

The lowest normalized Kendall-tau distance is between Metric 2 and Metric 3. This can also be seen in Table 4, where only R-SCG with Rank 6 and 7 are swapped.

5.2 Secure Coding Guidelines for C++

Table 3 shows the results for the C++ programming language of the ranked secure coding guidelines for metrics 1..4 and from the students' input. In this table, lower numbers mean higher ranks and larger numbers mean lower ranks. Table 5 shows the corresponding Kendall distance between the ranked lists.

Same as for the C secure coding guidelines, we can group the results into three clusters 1:$\{R^{(1)}_{\text{C++}}\}$, 2:$\{R^{(2)}_{\text{C++}}, R^{(3)}_{\text{C++}}, R^{(4)}_{\text{C++}}\}$ and 3:$\{P_{\text{C++}}\}$ according to their relative distances. For this case, the following results are immediately apparent: (1) the clusters are the same as for the C programming language, (2) there are three values which have the zero distance (i.e. are the same) and (3) the distance $\{P_{\text{C++}}\}$ to the other ranked lists is about the same as for the C R-SCG.

For the first observation, this re-states that the metrics 2, 3 and 4 do not produce significantly different results, as for the C SCG case. The second observation means that, for the C++ case, the metrics have lead to exactly the same R-SCG results (i.e. the same ranked list). The third observation means that the students, as for the C R-SCG, have a different perception for what is important as what was extracted from real-world data.

Furthermore, we see that the Top-7 R-SCG for C++ are all the same, independently of using Metric 1, 2, 3 or 4. The same cannot be said for the ranking obtained from the students.

5.3 Threat to Validity

We can see the following possible sources of threats to the validity of this work.

1. We have selected four metrics. However, we might have missed the definition of a metric that leads to very different results (maybe even close to the Students' ranking). However, our experience in the field tells us that the metrics hereby defined and the results obtained are consistent with what has been observed in practice.
2. Only 34 students have been involved in the questionnaire and the statistical results might differ if we increase the population size.
3. We have recurred to cybersecurity experts to untie SCG which had the same metric value. Holm et al. [14], and Allodi et al [1] discuss possible discrepancies that expert opinion might add to the scoring. Nevertheless, the results hereby presented have shown that the Kendall distance is not too much sensitive on the values of the scores.
4. Our work did not consider the impact of the standard deviation of the SCG metric. Taking this into consideration, the Kendall distance between the participants answers and the computed rankings could change and also lead to different conclusions.
5. This work did not consider SCG that can be checked with an automatic tool, such as static code analysis. However, our experience is that it is not sufficient to use tools, the developers should also know how to interpret their results. This is only possible with training and awareness.

6 Conclusions and Further Work

In an industrial context, training of software developers in secure coding is a costly activity that needs careful thought and planning. Software bugs often result in vulnerabilities which, when exploited, can lead to serious damage. Secure coding guidelines (SCG) exist nowadays in order to educate software developers and make them aware on how to avoid writing such bugs. However, it has been previously shown that not all software developers are knowledgeable on the said secure coding guidelines. Combined with the restrictions from the industry, this paper proposes a method to rank secure coding guidelines which in turn can be used to prioritize the training of software developers on the SCG. By focusing attention and addressing the most important secure coding guidelines(i.e. the ones that cause the most impact) first, a trained software developer can avoid the major problems and software bugs that have been plaguing the industry.

The major contribution of this work are two sets of ranked secure coding guidelines, one for C and another for C++. Another contribution of this work is the comparison of the gap between industry relevant ranking and ranking from academic students. Here we also see that, although academic students might even be well trained in writing secure code, their ranking of SCG did not match what is obtained from real-world data. As further work we would like to evaluate how and if the usage of automated tools (e.g. static code analysis) influences the results hereby presented.

References

1 Luca Allodi, Sebastian Banescu, Henning Femmer, and Kristian Beckers. Identifying relevant information cues for vulnerability assessment using cvss. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, pages 119–126, New York, NY, USA, 2018. ACM. doi:10.1145/3176258.3176340.

2 Motor Industry Software Reliability Association. Guidelines for the use of the c language in critical systems. Standard, Motor Industry Software Reliability Association, Nuneaton, Warwickshire, UK, March 2012.

3 Motor Industry Software Reliability Association. Additional security guidelines for misra c:2012. Standard, Motor Industry Software Reliability Association, Nuneaton, Warwickshire, UK, March 2016.

4 Roberto Bagnara, Abramo Bagnara, and Patricia Hill. The MISRA C coding standard and its role in the development and analysis of safety- and security-critical embedded software. *CoRR*, abs/1809.00821, 2018. arXiv:1809.00821.

5 International Electrotechnical Commission. Industrial communication networks - network and system security - part 2-1: Establishing an industrial automation and control system security program. Standard, International Electrotechnical Commission, October 2010.

6 International Electrotechnical Commission. Security for industrial automation and control systems - part 4-1: Secure product development lifecycle requirements. Standard, International Electrotechnical Commission, January 2018.

7 Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In *IEEE Symposium on Security and Privacy*, pages 121–136, San Jose, CA, USA, 2017. IEEE Computer Society. doi:10.1109/SP.2017.31.

8 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. Standard, NIST, March 2018.

9 Pascal Gadient, Mohammad Ghafari, Patrick Frischknecht, and Oscar Nierstrasz. Security code smells in android ICC. *CoRR*, abs/1811.12713:3046–3076, 2018. arXiv:1811.12713.

10 Tiago Gasiba, Kristian Beckers, Santiago Suppan, and Filip Rezabek. On the requirements for serious games geared towards software developers in the industry. In Daniela E. Damian, Anna Perini, and Seok-Won Lee, editors, *27th IEEE International Requirements Engineering Conference, RE 2019, Jeju Island, Korea (South), September 23-27, 2019*. IEEE, 2019. URL: https://ieeexplore.ieee.org/xpl/conhome/8910334/proceeding.

11 John Goodall, Hassan Radwan, and Lenny Halseth. Visual analysis of code security. In *Proceedings of the Seventh International Symposium on Visualization for Cyber Security*, VizSec '10, pages 46–51, New York, NY, USA, 2010. ACM. doi:10.1145/1850795.1850800.

12 Katerina Goseva-Popstojanova and Andrei Perhinschi. On the capability of static code analysis to detect security vulnerabilities. *Inf. Softw. Technol.*, 68(C):18–33, December 2015. doi:10.1016/j.infsof.2015.08.002.

13 Norman Hansch and Zinaida Benenson. Specifying it security awareness. In *25th International Workshop on Database and Expert Systems Applications, Munich, Germany*, pages 326–330, September 2014. doi:10.1109/DEXA.2014.71.

14 Hannes Holm and Khalid Afridi. An expert-based investigation of the common vulnerability scoring system. *Computers & Security*, 53, May 2015. doi:10.1016/j.cose.2015.04.012.

15 Siv Houmb, Virginia Franqueira, and Erlend Engum. Quantifying security risk level from cvss estimates of frequency and impact. *Journal of Systems and Software*, 83:1622–1634, September 2010. doi:10.1016/j.jss.2009.08.023.

16 Hongyi Hu, Bryan Eastes, and Michelle Mazurek. 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, MD, USA, August 2018.

17 Russell Jones and Abhinav Rastogi. Secure coding: Building security into the software development life cycle. *Information Systems Security*, 13(5):29–39, 2004.

18 Ankur Joshi, Saket Kale, Satish Chandel, and Dinesh Pal. Likert scale: Explored and explained. *British Journal of Applied Science & Technology*, 7:396–403, January 2015. doi:10.9734/BJAST/2015/14975.

19 Maurice Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1-2):81–93, June 1938. doi:10.1093/biomet/30.1-2.81.

20 Ryo Kurachi, Hiroaki Takada, Masato Tanabe, Jun Anzai, Kentaro Takei, Takaaki Iinuma, Manabu Maeda, and Hideki Matsushima. Improving secure coding rules for automotive software by using a vulnerability database. In *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–8, September 2018. doi:10.1109/ICVES.2018.8519496.

21 Na Meng, Stefan Nagy, Danfeng Daphne Yao, Wenjie Zhuang, and Gustavo Arango. Secure coding practices in java: challenges and vulnerabilities. In *IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 372–383, May 2018. doi:10.1145/3180155.3180201.

22 MITRE-Corporation. Common weaknesses enumeration, 2019. URL: https://cwe.mitre.org/.

23 MITRE-Corporation. CVE details, 2019. URL: https://www.cvedetails.com/.

24 Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, and Matthew Smith. Deception task design in developer password studies: Exploring a student sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 297–313, Baltimore, MD, 2018. USENIX Association. URL: https://www.usenix.org/conference/soups2018/presentation/naiakshina.

25 National Institute of Standards and Technology. Nist special publication 800-37, guide for applying the risk management framework to federal information systems a security life cycle approach. Standard, NIST, February 2010.

26 International Standards Organization. ISO/IEC 27002:2013 – Information technology – Security techniques – Code of practice for information security controls. Standard, International Standards Organization, October 2013.

27 Blerim Rexha, Arbnor Halili, Korab Rrmoku, and Dren Imeraj. Impact of secure programming on web application vulnerabilities. In *2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS)*, pages 61–66, November 2015. doi:10.1109/CGVIS.2015.7449894.

28 IEEE Spectrum. The Top Programming Languages 2018. https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018, 2019. [Online; accessed 27-October-2019].

29 PCI SSC. Payment Card Industry – Payment Application Data Security Standard – Requiremenst and Security Assessment Procedures, v3.1. Standard, PCI SSC, May 2015.

30 Madiha Tabassum, Stacey Watson, Bill Chu, and Heather Richter Lipford. Evaluating two methods for integrating secure programming education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education, SIGCSE 2018, Baltimore, MD, USA, February 21-24, 2018*, pages 390–395, 2018. doi:10.1145/3159450.3159511.

31 Carnegie Mellon University. Secure Coding Standards. https://wiki.sei.cmu.edu/confluence/display/seccode, 2019. [Online; accessed 19-March-2019].

32 Michael Whitney, Heather Richter Lipford, Bill Chu, and Tyler Thomas. Embedding secure coding instruction into the ide: Complementing early and intermediate cs courses with eside. *Journal of Educational Computing Research*, 56:073563311770881, May 2017. doi:10.1177/0735633117708816.

33 Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, September 2016. doi:10.1007/s11390-016-1672-0.

34 Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. Are code examples on an online q&a forum reliable?: A study of api misuse on stack overflow. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 886–896, New York, NY, USA, 2018. ACM. doi:10.1145/3180155.3180260.