6th International Conference on Higher Education Advances (HEAd'20)
Universitat Politècnica de València, València, 2020
DOI: http://dx.doi.org/10.4995/HEAd20.2020.11114

Teaching Security in Introductory C-Programming Courses

Dieter Pawelczak
Institute for Software Engineering, University of the Bundeswehr Munich, Germany.

Abstract

The challenges in the age of digitalization demand that universities qualify their computer science and engineering graduates well with respect to IT Security (information technology security). In engineering education such lectures are often offered as an elective subject, only. We propose to teach security aspects with respect to robustness and correctness already in the introductory programming course and therefore to cover at least parts of the overall field of IT Security as a compulsory subject for all students. The paper describes the integration of some rules and recommendations from the SEI Cert C Coding Standard into our introductory C programming course and discusses our experience with the course over the last two years with respect to its contents, realization, evaluation and examination.

Keywords: *Teaching Programming; IT Security; Cybersecurity.*

This work is licensed under a Creative Commons License CC BY-NC-ND 4.0
Editorial Universitat Politècnica de València

1. Introduction

In the context of digitalization, computer security is playing an increasingly important role. Since the turn of the millennium, the ACM (Association for Computing Machinery) and the IEEE (Institute of Electrical and Electronics Engineers) have been calling for computer science education to be further adapted to secure software development and cybersecurity, compare ACM and IEEE (2016). For computer science degrees, this topic is usually covered in advanced courses such as *IT Security* or *Secure Software Engineering*. With respect to engineering education, *IT Security* is often an elective course. Analysing the module descriptions for the undergraduate degree programs *Electrical Engineering and Information Technology* of 17 universities in Bavaria shows that only 4 programs request IT security as a compulsory subject in 2020. This represents less than 24 %. About 35 % of the university degree programs offer elective courses, resulting in 41 % not covering such subjects at all. Due to the significance of the topic Williams et al. (2014) suggest introducing security aspects already in introductory programming courses. This approach could simplify the integration of IT security aspects in Bachelor degree programs that currently lack such content. As our analysis relies on the module descriptions only, we expect that some introductory courses already embed these subjects without having mentioned it in their description.

We have integrated at our university IT security aspects with respect to robustness and correctness in our introductory C-programming course for our electrical engineering degree program in the last two years and propose the idea that describing security issus can arouse the students' interest and might even help with understanding the execution of a computer program. In this paper we describe our experiences with the course, present the students' evaluation of the course and discuss the assets and drawbacks.

2. Related Work

An extensive set of rules, examples of vulnerabilities and instructions on how to properly program in C with security awareness can be found in the SEI Cert C Coding Standard (2016) by the Software Engineering Institute of the Carnegie Mellon University. However, the security subjects have to be carefully chosen, to fit in with the scope of known concepts for the novice programmers. As Bandi et al. (2019) discuss, secure coding is often not covered by classical *IT Security* lectures. Therefore, dealing with security aspects in an introductory programming course cannot replace an advanced course on IT security; and vice versa. Subjects focus mainly on the aspects robustness and correctness, as required for secure coding and less on integrity or confidentiality, like e.g. cryptographic protocols, compare Williams et al. (2014). Another promising way to easily integrate IT security topics in the curriculum is a game based approach, compare e.g. Anvik et al. (2019).

Novice programmers often find unexpected ways to solve their programming issues. As Gómez-Martín et al. (2009) propose, teachers have to counteract the "but it works"-syndrome; meaning that some students fiddle through their programming assignments by trial and error and stop as soon as they think it fulfills the main requirements of the task. Such solutions are often open for many security issues, hard to maintain or to adapt. Applying security requires a more abstract and model-based thinking and to think outside the box. All of these competencies we would expect from students enrolled in engineering or computer science programs, but are rarely to be found, compare e.g. Zehetmeier et al. (2019).

The importance of teaching students programming with security awareness from the beginning is obvious, because it is difficult to adapt bad habits or to eliminate misunderstandings later. Furthermore, as Zhu et al. (2013) point out: many textbooks on programming provide little information on security or may even contain vulnerabilities. In addition, modern compilers print warnings about security issues, so students need to learn early how to deal with them. As compiler messages in general are "considered unhelpful" – compare Becker et al. (2019) – security diagnostics require further understanding.

Although previous research as e.g. by Williams et al. (2014) reports about successful integration of secure coding into introductory courses – even without changing the workload of the students, compare e.g. Bandi et al. (2019) – we have to keep in mind, that many students already struggle with programming itself and security aspects can also be seen as an add on. Still, IT security topics, cybersecurity, hacking competitions etc. arouse the interest of many students and might help to foster students' intrinsic motivation with respect to programming. We even propose that demonstrating security vulnerabilities might actually help with understanding the execution of a computer program.

3. Teaching Security within the Introductory C Course

Based on the SEI Cert C Coding Standard (SEI, 2016) we identified rules and recommendations, and derived use cases, that can easily be integrated into an introductory course. Exemplarily we describe two use cases we applied in the course and evaluated the results in the examination. We selected these two for the paper because both were tested in the course examinations. Overall, we addressed 17 rules of the standard in the lecture.

3.1. Use Case 1 – String Input Results in Buffer Overrun

Figure 1 shows a simple constructed example of a buffer overrun that bypasses a password check by overwriting the stack memory beyond the reserved memory for the user name. During the demonstration of the example it arouses the interest of students especially as we show an incomplete password check that will deny any user by just returning 0 (i.e. false), compare line 5 in Figure 1. Thus, the expectation is that the program will hang in an endless

loop requesting the proper password. As the program's output shows on the lower right in Figure 1, we can successful log in without entering any password by partly overwriting the contents of the variable checked because of an input beyond the reserved 12 characters. This results in skipping the while-loop and the password check in line 14 ff.

```c
#include <stdio.h>

char CheckUserAndPassWord(const char* user, const char* password) {
    /* ToDo: currently always fail ... */
    return 0;
}

int main(void){
    char checked = 0; /* we need to check initially all users */
    char userName[12] = "";
    char passWord[12] = "";
    printf("Please enter username: ");
    scanf("%s", userName);
    while(!checked) {
        printf("Please enter password: ");
        scanf("%11s", passWord);
        checked = CheckUserAndPassWord(userName, passWord);
    }
    printf("Welcome %s! Successfully logged in.\n", userName);
    return 0;
}
```

```
Please enter username: 1234567890123
Welcome 1234567890123! Successfully logged in.
```

Figure 1. Example erroneous program to demonstrate a buffer overrun and to explain the memory model of fixed-length string arrays in C. Screenshot of the programming environment (Virtual-C IDE) as shown in the lecture.

The example can serve to demonstrate multiple issues. The vulnerability of software and the requirement to properly handle user input are both obvious. We show this example when learning how to read strings from the console. It also demonstrates the memory model of C with respect to local variables and fixed length arrays: local variables are assigned to addresses in ascending (or descending, dependent on the compiler in use) order. Thus writing beyond the reserved memory space of variable userName will affect the succeeding variable checked. Last but not least, we identify the defective instruction due to a compiler warning according to SEI Cert rule FIO47-C (SEI, 2016) in line 13, which is found in the parameters to scanf(): we have to explicitly define the maximum length to read (compare line 16), which is 11 characters due to the array size and because scanf() will automatically add a terminating character; thus we also repeat how strings are stored in C.

3.2. Use Case 2 – Proper Use of Input and Output Parameters

In the context of passing arguments to a function by the use of pointers, we discuss the concept of input and output parameters. While the first is used to pass information to a function, output parameters allow a function to pass information back to the caller. As a rule, an input parameter must be declared as const in order to prevent the function from modifying the information. The standard library according to ISO C18 implements all function declarations accordingly, thus we can read from the signature of a function, which parameters are input and which are output parameters. Figure 2 shows a misuse of that rule: a function calculating the length of a character string will also modify the string; both the stack contents before and after the call to this defective function are illustrated.

```
#include <stdio.h>


int StrLength(char* input) {
    int count = 0;
    while (input && input[0]) {
        count++;
        input[0] = '?';
        input++;
    }
    return count;
}

int main(void){
    char modulName[] = "C-Programming";
    printf("Length of moduleName(=\'%s\') is: %d\n",
        modulName, StrLength(modulName));
    return 0;
}
```

a) Stack before call to StrLength()

b) Stack after call to StrLength()

```
Length of moduleName(='?????????????') is: 13
```

Figure 2. Example erroneous program to demonstrate wrong input parameter handling.

Although this example is obviously wrong, it shows that we should not call a function expecting an input parameter, which is not declared as constant. Especially as we usually use functions that were third party developed with no access to the source code. Declaring the parameter input as constant would lead to a compiler error and thus show the defective line of code (here line 8 which overwrites the string). As a side effect, we also learn about the parameter passing to a function: most students do not expect, that modulName has already been modified as the parameter seems to be passed to printf() before the call to StrLength() has been invoked.

4. Review of the Course

4.1. Examination

Adding security aspects to our introductory course had no directly measurable effect on the overall examination results. Still, the results were not inferior to the preceeding examinations, although additional subject matters were assessed in the examination: both use cases from Section 3 were expected to be handled properly and incorrect answers led to a lower final score. We analysed the impact on the examination results. For both course years, we took 81 exams into account; we excluded exams that left the corresponding tasks blank as we could not tell the reason (out of time or lack of knowledge). About 51 % of the students prevented a possible buffer overrun by limiting the input length (use case 1). Only one third of the students declared input parameters properly (use case 2). Interestingly, the percentage differs for students who failed the examinations 41 % of that group passed use case 2, while only 18 % passed use case 1. It is also noticeable, that students with good grades in particular answered these cases wrong, while the majority of students with

average grades answered them properly. An explanation could be that students with previous knowledge of C either found the security aspects less important, or that they did not attend the lecture and therefore couldn't achieve these points in the examination. That would confirm the statement of Zhu et al. (2013), that there is a lack of security awareness in existing programming courses and that textbooks still contain vulnerabilities.

4.2. Evaluation

Students gave us feedback with respect to secure coding on the following four questions, compare Figure 3:

A) Course contents regarding secure coding are very important.

B) Examples of security vulnerabilities deepens my understanding how C programs works.

C) Course content regarding secure coding complicate my understanding of C.

D) Compiler warnings on security issues are more distracting during programming.

We received feedback from 45 students. We also asked the students to self-assess their programming knowledge before the course and evaluated the results for two different groups: 16 % rated themselves as skilled programmers before the course (group 1), while 58 % have little or no previous knowledge (group 2).

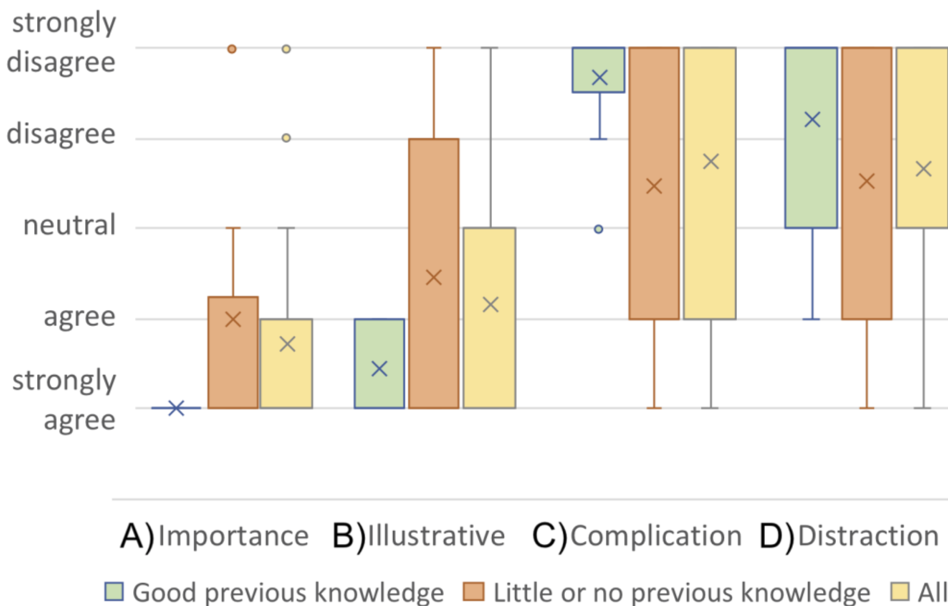

Figure 3: Results from the students' evaluation Questions A-D for the two groups and the whole class (5 grade Likert scale); the "x" represents mean values and "o" outliners.

Most students (83 %) were aware of the importance of security aspects in the C programming, compare Question A (Figure 3). Students with good programming knowledge (group 1) all agreed that examples of security vulnerabilities are helpful to explain how C programs are executed, compare Question B (Figure 3). The answer of the group 2 show a big deviation in their answer. Of course, knowledge on C and knowledge about possible vulnerabilities are closely interwoven: knowing the memory model of C allows to easily understand a buffer overrun. About 27 % of group 2 disagreed with our proposal that showing e.g. a buffer overrun helps to understand how C programs work. Still, 67 % of that group agreed with that. There is a big difference in answers between both groups with respect to Question C in Figure 3. While almost all students of group 1 disagreed that additional course content about secure coding complicates their understanding on C programming, about 33 % of the second group agreed. Some students even state in their evaluation, that this is an add-on, they have to learn for the examination. The answers to Question D showed the biggest deviation in both groups. Even some students from group 1 agreed that warnings about security issues distract them during programming. On the other hand, over 54 % of all students disagreed on this point. Becker et al. (2019) point out that more research is required for generating proper diagnostic messages by compilers, especially with respect to new learners. Future research should also include security related diagnostics.

4.3. Lecturer's Experience

Although the results in the examinations with respect to security issues were below the expectation of the lecturer, he gained positive experiences with these new subjects during the course. Showing short examples with surprising effects lightens the mood in the course and immediately initiates a discussion to debate several topics. As we found out in the first year, the examples need to be short and the surprising effect needs to be easy to grasp. If we expect too much knowledge about security, such examples will not have a positive effect even though they might be especially important for IT security. It is important to take enough time for the demonstration and the discussion of the examples. The best approach is to write the example live in the course, as students can follow the implementation better in such a reduced tempo and to stepwise fix the code to give an accurate solution. Otherwise it is hard for students to comprehend the meaning of the defective and the proper code and the learning effect is reduced to teacher's talk: "you should not do the following …".

5. Conclusion and Outlook

For two years we have been integrating security aspects into our introductory C programming course by discussing defective code snippets and correcting them, especially with respect to robustness and correctness. We see the need to foster the security awareness of our engineering students in order to prepare them for their future tasks in a more and more

digitalized world. We also found a lack in the engineering education in Bavaria with too little coverage of this topic in compulsory subjects and see our approach as one possible way to increase IT security awareness in engineering education. Properly selected examples like for instance the example of a buffer overrun (compare Section 3.1) can in addition to the security aspects also serve to enhance the understanding for the programming language; the majority of the course participants agreed on this in their evaluation feedback. From the lecturer's point of view, such demonstrations arouse the interest of students much easier compared to standard programming examples. In accordance with Bandi et al. (2019), we did not increase the workload for students in the course. But due to the importance of the topic and the positive experience, we plan to extend the weekly lecture hours for the course from 4 to 5 hours. This shall give more time to discuss these examples in more detail. Thus, all our engineering students get an introduction into computer security. Since 2019 we offer students having specialized in the field of applied computer technology an advanced compulsory module on computer security.

References

ACM and IEEE (2016). Computer Engineering Curricula 2016 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. *Tech. Rep.*, Association for Computing Machinery (ACM) and IEEE Computer Society.

Anvik, J., Cote, V., and Riehl, J. (2019). Program Wars: A Card Game for Learning Programming and Cybersecurity Concepts. In *Proc. of the 50th ACM Tech. Symp. on Comp. Science Education (SIGCSE '19), 393–399. doi: https://doi.org/10.1145/3287324.3287496*

Bandi, A., Fellah, A., Bondalapati, H., 2019. Embedding security concepts in introductory programming courses. *J. Comput. Sci. Coll*. 34, 4 (April 2019), 78-89.

Becker, B. A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D. J., Harrington, B., Kamil, A., Karkare, A., McDonald, C., Osera, P.-M., Pearce, J. L., and Prather, J. (2019). Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proc. of the Working Group Reports on Innovation and Technology in Comp. Science Education*, 177–210. doi: https://doi.org/10.1145/3344429.3372508

Gómez-Martín, M. A., and Gómez-Martín, P. P. (2009). Fighting against the 'But it works!' syndrome. In *Proc. XI Int. Symp. on Computers in Education*, SIIE'09 (Coimbra, Portugal)

SEI, 2016. SEI CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems. Online: https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=454220

Williams, K. A., Yuan, X., Yu, H., Bryant, K., 2014. Teaching secure coding for beginning programmers. *J. Comput. Sci. Coll*. 29, 5 (May 2015), 91–99

Zhu, J., Lipford-Richter, H., Chu, B., 2013. Interactive support for secure programming education. In *Proc. of the 44th ACM Tech. Symp. on Comp. Science Education* (SIGCSE '13), ACM, 687-692. doi: https://doi.org/10.1145/2445196.2445396

Zehetmeier, D., Böttcher, A., Brüggemann-Klein, A., and Thurner, V. (2019). Defining the Competence of Abstract Thinking and Evaluating CS-Students' Level of Abstraction. In *Proc. of the 52nd Hawaii Int. Conf. on System Sciences*, 7642-7651