# Entailing Security Mindset in Foundational CS Courses: An Interactive Approach

Vahab Pournaghshband
*Computer Science Department*
*University of San Francisco*
San Francisco, USA
vahab.p@usfca.edu

Hassan Pournaghshband
*Software Engineering Department*
*Kennesaw State University*
Atlanta, USA
hpournag@kennesaw.edu

***Abstract*—The integration of computer security courses into computing curricula has become standard practice. However, these courses are often deferred to later stages of the curriculum due to the perceived need for advanced computer science knowledge. This deferral perpetuates the lack of a security mindset, leading to overlooked security vulnerabilities in students' code. This study advocates for the early introduction of security-related courses, ideally in introductory computing courses, by proposing an interactive teaching approach to instill a security mindset in students. We present a method employing a security-oriented programming assignment graded primarily on understanding conventional computing concepts. This approach progressively introduces critical security principles and practices through the interactive assignment, inherently security-sensitive, involving the implementation of application programming interfaces for routine software operations. The paper outlines our approach's steps, discusses essential security concepts and their integration, and outlines an evaluation methodology to measure the effectiveness of our approach.**

***Index Terms*—Security education, computer security, security mindset**

## I. Introduction

The rapid integration of software into various aspects of modern life has been accompanied by a surge in cyber threats and malicious activities. Hackers have significantly enhanced their capabilities to exploit software vulnerabilities, resulting in substantial disruptions to technological infrastructures. The annual cost of global cybercrime is now estimated to be $600 billion, up more than $100 billion from 2016 [1]. Recent incidents, such as the massive data breach affecting 37 million T-Mobile customers and the extensive downtime of the Pacific island nation of Vanuatu's government digital networks, underscore the criticality of addressing software security [2], [3]. Moreover, the intersection of geopolitics and the cyber-attack landscape, as evident in conflicts like the Russia-Ukraine war, reinforces the necessity for proactive cybersecurity measures [4].

Software security vulnerabilities are the most common cause of software security breaches [5]. As highlighted in [6], simply relying on security patch management and distribution to fix these bugs could present challenges. This makes early detection of security bugs in the pre-release versions of programs an effective strategy [5]. Although there are various reasons for a program to fail and the programmers' inability to anticipate how these failures can be potentially exploited, the absence of a security mindset significantly hinders the early detection of exploitable bugs in software development. Bruce Schneier defines the security mindset in the following excerpt from his blog [7]:

> "Security requires a particular mindset. The security mindset involves thinking about how things can be made to fail. It involves thinking like an attacker, an adversary or a criminal. You don't have to exploit the vulnerabilities you find, but if you don't see the world that way, you'll never notice most security problems."

Teaching early programmers this security mindset is a pivotal step toward reducing cyberattacks and enhancing the security of future technological systems. Therefore, it is crucial for students to acquire the necessary skills and knowledge early in their academic programs to effectively handle these problems. Although learning the security concepts generally requires a more advanced knowledge of computer science, learning the security mindset, with careful planning and preparation, is feasible within foundational programming courses. While the primary focus of introductory programming courses is on foundational programming concepts, integrating teaching the security mindset into these courses is valuable [8].

In this paper, we propose an interactive approach to promote the security mindset among students in introductory computer programming courses. We outline a methodology to develop and teach this mindset within such a course, utilizing carefully crafted programming assignments that align with intended introductory course objectives. The core idea of our approach is to utilize a programming assignment with this matter in mind while not compromising other required subjects in the course. This assignment should be security-sensitive by nature, with application programming interface (API) implementation for the corresponding software that performs routine operations expected from such software system. Through our proposed process, at different steps, the students are introduced to major security concepts and practices. The assignment should be merely graded based on students' understanding of only conventional computing concepts.

This paper is organized as follows: Section II describes

1) Release the assignment specification to students. Students implement the assignment according to the given specifications.
2) Instruct students on exploiting inherent security vulnerabilities in their code, causing program crashes or undesired behavior.
3) Facilitate a discussion forum with students to explore their understanding of the program's failure and identify the parts of their code responsible for the abnormal behavior.
4) Educate students on the security consequences stemming from the identified vulnerabilities, enhancing their awareness of security issues.
5) Conduct a class discussion on effective strategies to mitigate or eliminate these vulnerabilities in their code.
6) Conclude the process by presenting and demonstrating widely recognized practices for mitigating or eliminating these vulnerabilities.

Fig. 1: Teaching the Security Mindset: A Structured Process.

the existing literature. In Section III, we provide a detailed description of our approach and discuss how it can be effectively employed in an early computer programming course. We explain the characteristics of a suitable programming assignment for our approach, along with an explanation of each of the steps of the process. We demonstrate how we can effectively teach basic programming concepts, which is the primary goal of any programming course, while developing the necessary security mindset for the students. We also show how our approach which is naturally interactive plays a significant role in engaging our students in the whole process, which ultimately helps them to benefit from it and appreciate the notion of computer security. Section IV provides an example of such an assignment. In Section V, we outline an evaluation methodology to assess the effectiveness of this approach. Finally, Section VI concludes the paper.

## II. Related Work

The issue of teaching the security mindset in introductory courses has been examined first in [8]. In [8], we demonstrated how to teach the security mindset through a set of carefully crafted examples in lectures. Extensive literature advocates for incorporating security concepts into early programming courses [9]–[11], with an increasing focus on integrating security education into undergraduate computer science curricula [12]–[14]. While some suggest a comprehensive approach of teaching security concepts within a single course [13], others recommend effective track approaches involving a sequence of specialized security-focused courses [14]. Authors in [8], [15] argue not only for the inclusion of the security mindset in foundational computing courses but also for its continuous emphasis throughout the student's undergraduate program. Other studies propose introducing security concepts in upper-division courses by augmenting security design components into existing projects [16].

Various studies have also dealt with the challenge of incorporating security education into the undergraduate curriculum of computer science or related disciplines [13]. In response to the growing demand for cybersecurity education, a range of educational initiatives has emerged. Notably, the ACM/IEEE computing curricula were updated in 2013 to incorporate cybersecurity [17]. Building upon this, the Joint Task Force on Cybersecurity Education (JTF) released extensive curricular guidance [18] to assist educators in developing cybersecurity courses.

## III. Methodology

In Section I, we emphasized the importance of computer security and the role of vigilant programmers in this domain. Furthermore, in Section II, we highlighted the widespread recognition of the need to integrate computer security education into the established curriculum for computer security. Therefore, it is crucial for computing students to have a foundational understanding of computer security as early as possible, aligning with the existing institutional curriculum. In this section, we present our methodology and demonstrate how to teach this security mindset in an interactive fashion, which has proved to be very effective in fostering our students' curiosity and interest in this critical domain.

The main idea behind our approach is to present the students with an assignment evaluated only based on their understanding of traditional computing concepts. But through a process, at different steps, the assignment strategically introduces them to major security concepts and practices. To illustrate our approach, we focus on the "Banking Software" assignment as a representative example, which has proven successful in our courses. Importantly, any analogous assignment with inherent security sensitivities would yield similar outcomes. The Banking Software project involves the development of an API for software handling routine banking transactions like deposits, withdrawals, account inquiries, new account creation, and transaction displays—operations that inherently require a security-sensitive approach. Fig. 2 provides the list of the functions students are tasked with implementing. Through a process, the students are introduced to crucial concepts such as one-way hash functions and symmetric encryption. They are provided with blackbox APIs to execute vital security operations, employing SHA-1 for one-way hash functions and AES for symmetric-key encryption.

```
bool makeDeposit(const char account_no[], const char passphrase[], double amount)
bool makeWithdrawal(const char account_no[], const char passphrase[], double amount)
double getBalance(const char account_no[], const char passphrase[])
void printTopTenTransactions(const char account_no[], const char passphrase[])
bool addNewAccount(const char account_no[], const char passphrase[])
bool log(ofstream& log_file_stream, const char log_message[])
```

Fig. 2: Required utility functions for the banking software.

The rationale behind our interactive teaching approach is to actively engage students through a series of discussions and debates centered around *their own* submitted code. Fig. 1 illustrates our six-step process, each accompanied by a brief description.

## IV. An Example: Banking Software

In this section, we present an assignment we used to implement our proposed interactive approach—Banking Software. The programming language used in our introductory Computer Science courses is C. This assignment can be used as an example to incorporate the six-step process proposed in this paper.

In the assignment specification, we first include the definition of security tools used in the assignment (i.e., encryption and one-way hash functions), which also includes a brief overview of how they will be used in the implementation of the banking software. This is a challenging step for instructors because, besides introducing the security concepts and tools to their students, they need to educate them on how those concepts and tools should be applied to what they are implementing. The overall design of the system is presented in Fig. 3. In this part of the assignment specification, the students will learn about the overall design of a secure system and how to apply the security concepts they just learned in the implementation of the system. The next part of the assignment explains what the students are to implement thoroughly. Fig. 2 lists the utility functions the students are to implement for this assignment. In the last part of the assignment, students are introduced to software testing, which requires writing a set of unit tests to test their implemented functions. They are expected to come up with test cases and descriptive scenarios. Students' submissions for the testing component of the assignment will also be used in assessing our methodology (Section V).

The remaining sections present the security concepts introduced and discussed in banking software.

### A. Buffer Overflow

In this particular segment, students are tasked with inputting a significantly lengthy passphrase. Given that the assignment inherently involves receiving user input on various occasions, requesting students to provide a passphrase is a standard request. However, considering a scenario where the user's passphrase is stored as a C string, a potential buffer overflow vulnerability exists within the students' code. Requiring students to input an exceedingly long passphrase can cause their programs to crash. While a program crash is an undesirable outcome, it pales in comparison to the potential catastrophic consequence of a malicious user exploiting this vulnerability to gain complete control over the host machine running the vulnerable code. It is crucial to explain to the students that a program crash, in this context, represents a best-case scenario.

In this context, emphasis is placed on input validation and the utilization of secure functions, both being common practices to mitigate buffer overflow vulnerabilities. Students are guided to understand the importance of robust input validation techniques and the use of secure functions to eliminate the risk of buffer overflows, thereby fortifying their code against potential security threats.

### B. Denial of Service Attack

In the assignment, critical account information is stored in an encrypted file linked to each user account, which the program accesses to retrieve data. However, a potential vulnerability arises if a malicious entity tampers with the corresponding file associated with a specific user, deliberately inflating its size substantially. This can result in a denial-of-service (DoS) attack when the program tries to read from the corrupted file. The primary concern here is resource exhaustion, particularly regarding common readline APIs (e.g., `getline`), which are susceptible to DoS attacks when dealing with excessively large files.

Here, the students are guided to incorporate a file size check before reading from the file, thereby mitigating the potential for such attacks.

### C. Integer Overflow

In the assignment, students are instructed to use a dynamically created array to store the transaction history for each account. However, a critical vulnerability arises if the number denoting the size of the array at the time of creation, representing the total transactions, is too large. In such a scenario, the integer variable responsible for holding this value (expected array size) may overflow. This overflow can lead to the allocation of an insufficient memory buffer to accommodate the entire transaction history. Subsequently, when the array is populated with transactions, data might be written beyond the buffer's end. This could potentially enable the

**Overall Design:**

You are to write six utility functions for banking systems to use that follow a specific design. In this design, every bank account is associated to a file that holds information about the bank account's transaction history as well as some sensitive personal information about the account's holder (including the passphrase required for authorizing any transactions on that account). Every bank account is in this format "XXXXX-XXXXX", where X's are digits. For example, "12345-67890" is a valid account number, while "1234567890" and "ab234-56789" are not. Each bank account file follows the following format:

```
passphrase
account holder's personal information\n
transaction 1\n
transaction 2\n
...
transaction n\n
<EOF>
```

Since the machine that is designed to hold this highly sensitive information could potentially be vulnerable to attacks by hackers, we need to augment our system with some security. We use both encryption and one-way hash functions to improve the security of our system.

Every bank account file is encrypted by an independent passphrase, specified by the account holder at the account creation time. Note that we assume that the passphrase cannot contain white spaces (' ', '\t', or '\n'). For every transaction to take place, the following steps should take place in order:

1) The file should be decrypted.
2) It should check if the passphrase used to decrypt the file was valid. Technically, matching it with the first line of the account file should confirm that.
3) Make the transaction.
4) Encrypt back the file using the same passphrase.

This certainly adds some security to our system, since the content of the files are scrambled. So any hacker that breaches into the system should only see a bunch files with useless scrambled content. That is not entirely the case, if we leave the account numbers on the file names. To protect the account numbers from the hackers, we need to scramble them as well. However, we cannot simply encrypt them, as any encryption requires a key. That is where one-way hash functions can become useful. By making each account file name to be the MD5 hash of the account number, we can protect the set of account numbers while accessing the right file by just calculating the hash value of the account number. Remember that one-way hash functions are designed to be irreversible, hence knowing the hash values is worthless information. Below is an example of a file for bank account "12345-67890" (note that negative values indicate a withdrawal transaction):

Filename: d32fe4f1ca27cc145e359bdf0c8a3797

```
mypassphraseisweak
Alan Smith, SS#: 111-11-1111
0
20
100
-10
100
-200
```

One scenario that could describe this transaction history is that Alan wanted to withdraw $200 after making deposits $20 and $100, but since it was insufficient, a $10 overdraft penalty was applied to his account. He then deposits $100 before withdrawing $200 from his account some time later.

Fig. 3: Overall design of the banking software as presented in the assignment specification.

execution of malicious code residing in attacker-controlled memory, posing a significant security threat.

In this case, the students are introduced to the importance of implementing proper checks and mechanisms to handle potential integer overflow scenarios.

### D. Memory Leak

The assignment guides the students to allocate memory dynamically at different points in their program, such as when reading a particular user's transaction history. However, it is crucial to emphasize the proper deallocation of dynamically allocated memory. Failure to free memory appropriately can result in a memory leak. A memory leak can be exploited by an attacker to launch a DoS attack, intentionally triggering the leak. This can cause the program to exhibit unexpected behavior due to low memory conditions. Hence, educating the students to correctly handle and deallocate dynamically allocated memory is important to mitigate the potential vulnerability to DoS attacks.

### E. Clear Memory before Deallocating Memory

Memory allocated throughout a program's runtime often holds highly sensitive data, including users' personal information, passwords, and cryptographic keys, as demonstrated in the context introduced in our programming assignment. Access to this information by unauthorized processes running on the same system is a significant security concern. When memory is deallocated or freed, the freed memory chunk can be allocated to other processes, potentially exposing old data stored in memory. It is essential to highlight to the students that certain languages like C/C++ do not automatically clear the memory that a process frees, primarily for performance and efficiency reasons. Consequently, the responsibility of securely clearing memory that contains sensitive information rests with vigilant programmers.

In this assignment, students are first instructed that using `memset()` to fill the allocated memory with zeros just before freeing the memory is ineffective. Compiler optimization can remove this statement during the dead code elimination process. Subsequently, students are introduced to common data-scrubbing practices aimed at preventing the leakage of sensitive information in this context. These practices include the utilization of functions like `memset_s()` and `explicit_bzero()` to securely clear memory, ensuring that sensitive data remains protected even after deallocation.

## V. Evaluation Methodology

In this Section, we outline how we plan to evaluate our proposed methodology. We are currently conducting a longitudinal study over multiple offerings of the course. We will then analyze the collected data, and share and publish the results. To assess the effectiveness of our approach, we employ the following measures:

1) Measuring students' attitudes towards computer security
2) Students' test scenarios
3) Examining log file
4) Surveying the pleasantness of the experience

In the remaining part of this section, we present the details of each of the above measures.

### A. Measuring Students' Attitudes Towards Computer Security

We measure attitudes and behaviors toward cybersecurity prior to and after the assignment to see if their attitudes changed positively or negatively (or not at all) toward computer security. We accomplished this through pre-/post- test questionnaires to measure students' attitudes towards computer security. The format of this questionnaire is inspired by those used in other disciplines [19]. The questionnaire was given to the students before the assignment specification was presented to them, and a post-test was conducted after all discussions regarding the assignment were concluded. The purpose of this questionnaire is to identify students' perceptions and attitudes toward computer security. These questions are a number of Likert scale (strongly agree, agree, neutral, disagree, and strongly disagree) and some yes/no questions to evaluate the individuals' confidence, interest, usefulness, and professional constructs. Responses are converted to numerical scores for later analysis.

Below are sample questions in the questionnaire:

- *I am eager and willing to understand computer security concepts.*
- *I value understanding the rationale behind cyber attacks and defense.*
- *I am confident I can learn to understand computer security concepts.*

### B. Students' Test Scenarios

The assignment consists of two phases: implementation and testing. In the assignment specification for the second phase, we describe "Software Testing" and introduce the students to fundamentals concepts and practices in this domain such as like unit testing and integrate testing. The following paragraph is an excerpt from the specification:

> "There might be features that are not supported by the current design, but if you feel that given such API for developing a banking system, you expect certain situations to be handled in certain ways, then we would like to hear them from you, so include them in your test cases as well. Keep in mind, the purpose of testing is to ensure that the program behaves correctly and rather predictably in all cases, not just most cases! It is good coding and testing practice to come up with all possible scenarios and identify even those that the program will not behave as expected, even if we do not provide a fix for that now (perhaps because we have not yet learned how to). So, include such scenarios in your test file as well."

As part of our evaluation methodology, we examine the students submitted test scenarios carefully and check specifically if they include scenarios where the input from users are not in an expected format, more specifically seemingly malicious.

> Logs provide a way to monitor your systems and keep a record of security events, information access and user activities. Log everything that you feel is important and seem critical to you for a log auditor (or system administrator or a forensic analyst) to examine in critical situations. Also, overlogging could be problematic too, since it hides useful log entries among a bunch of stuff that is of no importance or interest. We leave that almost entirely up to you, what to log and how (in what format) you log these incidents. There is only one situation that we mentioned explicitly in this specification that you should log, but nevertheless there are more instances that you may consider logging.

Fig. 4: Introducing students to logging in the assignment specification.

### C. *Examining the Log File*

As part of the assignment, the students are introduced to the importance of logging and tasked to log suspicious events (Fig. 4). We then examine students' submissions to see what information they included in the log file. More specifically, we look for scenarios in which their programs generate log entries and how they log those events. For instance, are they considering multiple failed logging attempts as suspicious activity and log this event appropriately?

Furthermore, Log files are known to be attractive to malicious entities since they are often in plaintext. Therefore, we also examine if the generated log entry includes any sensitive information in its entirety (e.g., whether the actual corresponding account number is logged or not), or if the student was more vigilant about what information is being included in the log file (e.g., the leading digits of the account number is redacted in the log file).

### D. *Pleasantness of Experience*

The primary objective of the introductory course still is to emphasize learning the basic concepts of programming. By incorporating simple attack/defense scenarios to teach and verify program correctness interactively, we make learning those concepts more attractive to our students.

As stated by Fanelli et al. [20], "Security can make the other stuff more interesting. Studying security can lead students to a deeper understanding of computer science and information technology concepts. In many cases a thorough understanding of how a program works is needed to effectively attack or defend it." Therefore, we ask the students about about their perceptions of the assignment and the proposed methodology—whether it was an enjoyable and engaging experience. We use a scale ranging from 1 (low) to 5 (high) in this survey.

## VI. Conclusions

We advocate for integrating computer security into early computer programming courses—an idea with substantial potential. While acknowledging the challenges for both students and instructors, we firmly believe that the significant benefits for students outweigh these hurdles. In this paper, we outlined a six-step process to teach the security mindset interactively. Additionally, we introduced an ongoing evaluation methodology, incorporating pre- and post-test questionnaires, along with a thorough examination and analysis of students' test scenarios and log implementations.

## References

[1] McAfee, "Annual report on economic impact of cyber crime." [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/restricted/rp-economicimpact-cybercrime.pdf

[2] U. TODAY, "In latest t-mobile hack, 37 million customers have personal data stolen, company says." [Online]. Available: https://www.usatoday.com/story/tech/2023/01/20/tmobile-data-hack-37-million-customers/11088603002/

[3] NPR, "The pacific island nation of vanuatu has been knocked offline for more than a month." [Online]. Available: https://www.npr.org/2022/12/06/1140752192/the-pacific-island-nation-of-vanuatu-has-been-knocked-offline-for-more-than-a-mo

[4] NPR, "Russia bombards ukraine with cyberattacks, but the impact appears limited." [Online]. Available: https://www.npr.org/2023/02/23/1159039051/russia-bombards-ukraine-with-cyberattacks-but-the-impact-appears-limited

[5] J. Heffley and P. Meunier, "Can source code auditing software identify common vulnerabilities and be used to evaluate software security?" in *Proc of Annual International Conference on Systems Sciences*, 2004.

[6] H. Cavusoglu, H. Cavusoglu, and J. Zhang, "Security patch management: Share the burden or share the damage?" *Management Science*, vol. 54, no. 4, pp. 657–670, 2008.

[7] B. Schneier, "Schneier on security: The security mindset," https://www.schneier.com/news/archives/2016/02/bruce_schneier_the_s .html, 2008.

[8] V. Pournaghshband, "Teaching the security mindset to CS1 students," in *Proceeding of the 44th ACM technical symposium on computer science education*, ser. SIGCSE '13. ACM, 2013.

[9] T. Dimkov, W. Pieters, and P. Hartel, "Training students to steal: a practical assignment in computer security education," in *Proceedings of the 42nd ACM SIGCSE*, 2011, pp. 21–26.

[10] T. Kohno and B. D. Johnson, "Science fiction prototyping and security education: cultivating contextual and societal thinking in computer security education and beyond," in *Proc of ACM SIGCSE*. ACM, 2011.

[11] V. Pournaghshband and H. Pournaghshband, "Teaching security notions in entry-level programming courses," in *Proceedings of IEEE International Conference on Engineering, Technology, and Education*, 2021.

[12] K. Nance, "Teach them when they aren't looking: Introducing security in cs1," *IEEE Security Privacy*, vol. 7, no. 5, pp. 53–55, Sept 2009.

[13] L. F. Perrone, M. Aburdene, and X. Meng, "Approaches to undergraduate instruction in computer security," in *Proceedings of the American Society for Engineering Education Annual Conference and Exhibition*, 2005.

[14] S. Azadegan, M. Lavine, M. O'Leary, A. Wijesinha, and M. Zimand, "An undergraduate track in computer security," *SIGCSE Bull.*, vol. 35, no. 3, pp. 207–210, Jun. 2003.

[15] K. Nance and B. Hay, "Computer security-focused programming assignments in foundational cs courses," 2009.

[16] V. Pournaghshband and H. Pournaghshband, "Appending security theories to projects in upper-division cs courses," in *Proc of International Conference on Computer Science and Information Technology*, 2021.

[17] ACM Joint Task Force on Computing Curricula and IEEE Computer Society, "Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science," 2013.

[18] Joint Task Force on Cybersecurity Education, "Cybersecurity curricular guideline," *Computers & Education*, 2017. [Online]. Available: http://cybered.acm.org/

[19] N. F. Tahar, Z. Ismail, N. D. Zamani, and N. Adnan, "Students' attitude toward mathematics: The use of factor analysis in determining the criteria," *Procedia-Social and Behavioral Sciences*, vol. 8, 2010.

[20] R. L. Fanelli and T. O'Connor, "Experiences with practice-focused undergraduate security education." in *Proceedings of the 3rd international conference on Cyber security experimentation and test (CSET'10)*, 2010.