Apertem os Cintos... o Copiloto Sumiu! O Impacto do Ensino de Programação Segura em Computação

Nadia Luana Lobkov[1], **Paulo Ricardo Lisboa de Almedia**[1], **André Ricardo Abed Grégio**[1], **José Alexandre D'Abruzzo Pereira**[2]

[1] Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

[2]Faculdade de Ciência e Tecnologia – Universidade de Coimbra
Coimbra – Portugal

{nadialobkov, paulorla, gregio}@ufpr.br, josep@dei.uc.pt

Abstract. *Courses focused on computer programming are fundamental components of computing curricula. However, many programs do not adequately emphasize secure programming techniques, resulting in professionals who, although capable of developing software, do not do so securely. In this study, 85 questionnaires completed by students from two computing programs were analyzed. The students were divided into two groups: those who considered themselves capable of secure programming and those who did not. The results are concerning, revealing no significant difference in secure programming ability between the two groups. Additionally, the performance of students who received basic programming instruction with the simultaneous introduction of secure programming techniques was compared to that of students who were taught programming alone. Once again, no significant differences were observed, suggesting that teaching secure programming techniques during the initial stages of learning to program may be insufficient, or inadequate, for students to effectively acquire these skills.*

Resumo. *Disciplinas voltadas à programação de computadores são fundamentais nos cursos de computação. No entanto, muitos currículos não enfatizam adequadamente técnicas de programação segura, formando profissionais que, embora capazes de desenvolver software, não o fazem de maneira segura. Neste trabalho, foram analisados 85 questionários respondidos por estudantes de dois cursos da área. Os alunos foram divididos em dois grupos: aqueles que se consideravam aptos a programar com segurança e aqueles que não se consideravam capazes. Os resultados são preocupantes, revelando que não há diferença significativa na capacidade de programação segura entre os grupos. Além disso, comparou-se o desempenho de estudantes que tiveram aulas de programação básica com introdução simultânea de técnicas de programação segura com o de estudantes que cursaram apenas a programação convencional. Novamente, não foram observadas diferenças significativas, indicando que o ensino de técnicas de programação segura durante o aprendizado inicial de programação pode não ser suficiente, ou mesmo adequado, para promover o domínio dessas práticas.*

DOI: 10.5753/sbie.2025.12961

1. Introdução

Programação de computadores é uma disciplina fundamental na ciência da computação e um dos primeiros assuntos aos quais os estudantes deste tipo de curso aprendem. O ensino da programação, entretanto, enfatiza funcionalidades das linguagens em detrimento da segurança de código [McGraw 2006]. Com o aumento de ataques cibernéticos e vulnerabilidades em sistemas críticos, a integração de práticas de codificação segura na formação de desenvolvedores tornou-se essencial. Porém, muitos cursos de programação em nível superior ainda negligenciam esse aspecto, formando profissionais que reproduzem falhas conhecidas, como *buffer overflows*, *SQL injection* e *race conditions* [Howard and LeBlanc 2003].

Há mais de duas décadas a literatura da área vem mostrando que a maioria das vulnerabilidades comumente exploradas em sistemas poderia ser evitada com técnicas básicas de programação defensiva [Viega and McGraw 2001, Seacord 2013], além de argumentar que a segurança deve ser incorporada no ciclo de desenvolvimento desde o primeiro dia, não como um acréscimo após a liberação (*deployment/release*) do software [Barnum and McGraw 2005]. O CERT *Secure Coding Standards*, desenvolvido e publicado pelo *Software Engineering Institute (SEI)*, enfatiza a importância de se ensinar desde cedo práticas como validação de entrada, gerenciamento seguro de memória e princípios de mínimos privilégios [CERT/SEI 2024].

O ensino tradicional de programação, focado apenas em sintaxe e algoritmos, cria uma lacuna perigosa: alunos aprendem a escrever código que funciona, mas não código que resiste a explorações [Lam et al. 2022]. Além disso, o uso massivo de ferramentas de auxílio à codificação, como o Microsoft Copilot e o ChatGPT, propaga entre os estudantes a prática de copiar e colar funções e até mesmo programas completos sem uma análise mais minuciosa acerca da segurança do código gerado [Rahman et al. 2019, Hong et al. 2021].

Esse tipo de prática é especialmente crítica em linguagens como C e C++ que, se por um lado são flexíveis e dão grande poder ao programador, também cobram a responsabilidade de se saber o que se está programando. Nas linguagens citadas, erros de manipulação de memória são comuns e potencialmente catastróficos, onde inúmeros casos amplamente difundidos mostram que más práticas levaram a falhas de segurança amplamente exploradas [Votipka et al. 2020].

Considerando (i) a situação atual de ensino de programação em uma universidade brasileira, (ii) que alguns alunos foram expostos à conceitos de programação segura durante as disciplinas de programação introdutórias, e (iii) a auto-declaração dos estudantes sobre seus conhecimentos sobre segurança na codificação, o presente trabalho foca em responder às seguintes Questões de Pesquisa (QPs):

- **QP1**: Há diferenças entre alunos que se consideram capazes de programar de forma segura e aqueles que não se consideram?
- **QP2**: Os alunos que tiveram conceitos de programação segura enquanto aprendiam a programar, programam de maneira mais segura do que os demais?
- **QP3**: Os alunos melhoram a segurança de suas programações conforme progridem no curso, ou existe um plateau em que os alunos deixam de progredir?

Para responder tais perguntas, foram analisados os questionários de 85 alunos de

dois cursos de computação da Universidade Federal do Paraná. Os resultados mostram que o emprego de metodologias que tentam ensinar programação segura enquanto os alunos ainda estão aprendendo a programar podem ser insuficientes. Além disso, tais metodologias podem criar a falsa percepção nos estudantes de que eles sabem programar seguramente, mostrando indícios do Efeito Dunning-Kruger [Kruger and Dunning 1999], onde as lacunas de conhecimento dos estudantes podem ser impeditivos para que eles percebam os pontos em que precisam melhorar.

O restante do artigo está organizado da seguinte forma: na Seção 2 são apresentados os trabalhos relacionados. Já na Seção 3 é detalhada a metodologia usada para coleta e análise dos dados. Na Seção 4 são apresentados e discutidos os resultados. Finalmente, na Seção 5, são apresentadas as considerações finais e propostos os trabalhos futuros.

2. Trabalhos Relacionados

[Taylor et al. 2013] discutem a crescente importância do ensino de programação segura na formação em Ciência da Computação, destacando iniciativas que integraram "Segurança e Garantia da Informação"como uma área de conhecimento essencial. Os autores desmistificam concepções equivocadas, como a ideia de que não há espaço no currículo para programação segura ou que apenas ensinar técnicas de codificação segura resolverá os problemas de segurança de software. Além disso, são apresentadas ferramentas e projetos para facilitar a inclusão de práticas de programação segura em disciplinas introdutórias e avançadas. Por fim, é enfatizada a necessidade de envolver toda a comunidade acadêmica, desde professores até alunos, para criar uma “mentalidade de segurança” e melhorar a qualidade do software desenvolvido.

[Chi et al. 2013] propõem um framework para integrar práticas de codificação segura no ensino de programação, utilizando ferramentas de análise estática (como *FindBugs* [Pugh and Loskutov 2015] e *Splint* [Evans and Larochelle 2002]) em um sistema web chamado TSCPSTEM, o qual oferece exercícios práticos, quizzes e demonstrações adaptados a diferentes disciplinas (Ciências Naturais, Engenharia, Matemática e Ciência da Computação). O experimento com 70 estudantes de STEM mostrou um aumento de 130% no desempenho em testes após o uso dos módulos, comprovando a eficácia da abordagem. Os resultados destacam a importância de ensinar segurança desde o início da formação, preparando futuros desenvolvedores para evitar vulnerabilidades comuns, como buffer overflows e SQL injection.

[Singleton et al. 2020] abordam o problema das práticas de programação inseguras, especialmente no uso incorreto de primitivas criptográficas, como o armazenamento de dados sensíveis em texto puro em aplicativos bancários móveis, destacando a falta de integração desses conceitos em programas acadêmicos e a carência de ferramentas automatizadas para orientar desenvolvedores na escrita de código seguro. Para suprir tal lacuna, o artigo apresenta *CryptoTutor*, uma ferramenta que identifica automaticamente erros comuns em implementações criptográficas e sugere correções. Além disso, discutem como ferramentas como essa podem ser incorporadas em cursos de programação em níveis universitário e pré-universitário, visando melhorar a segurança do software desde a fase educacional.

[Espinha Gasiba et al. 2023] exploram o uso do ChatGPT como ferramenta para auxiliar desenvolvedores na identificação e correção de vulnerabilidades em código,

baseando-se em experimentos com cinco snippets vulneráveis em C/C++. Os resultados mostram que o ChatGPT identificou corretamente 60% das vulnerabilidades e sugeriu soluções criativas, porém com limitações como falta de contexto, alterações indesejadas na semântica do código e complexidade excessiva. O estudo destaca o potencial do modelo como auxílio educacional e profissional em segurança de software, mas alerta para riscos como plágio e dependência de dados desatualizados.

[Lam et al. 2022] investigam as lacunas no conhecimento de programação segura entre estudantes de Ciência da Computação, por meio de entrevistas com 21 alunos de duas universidades dos EUA. Os resultados revelaram que os alunos enfrentam dificuldades em áreas fundamentais, como compreensão de mensagens do compilador, uso de recursos online (como Stack Overflow), conhecimento de memória e funções inseguras em C (por exemplo, `gets`, `strcpy`), além de uma abordagem pouco crítica em relação à segurança. O trabalho destaca que cursos introdutórios frequentemente negligenciam práticas seguras, e que materiais didáticos e exemplos em sala podem reforçar más práticas. Esses achados reforçam a necessidade de integrar segurança de forma transversal no currículo, desde disciplinas iniciais, e de desenvolver intervenções educacionais que abordem essas lacunas, como módulos específicos ou ênfase em análise de código. O estudo complementa pesquisas anteriores sobre a eficácia de ferramentas de revisão de código e a importância de uma "mentalidade de segurança" desde o início da formação.

3. Metodologia

Para a realização deste estudo, foram aplicados questionários aos alunos de dois cursos de graduação do em Computação da Universidade Federal do Paraná. Os questionários foram amplamente divulgados aos alunos desses cursos, sendo que o preenchimento do questionário foi realizado de forma voluntária. A metodologia seguida para a criação dos formulários, aplicação, coleta e análise dos dados pode ser vista na Figura 1.

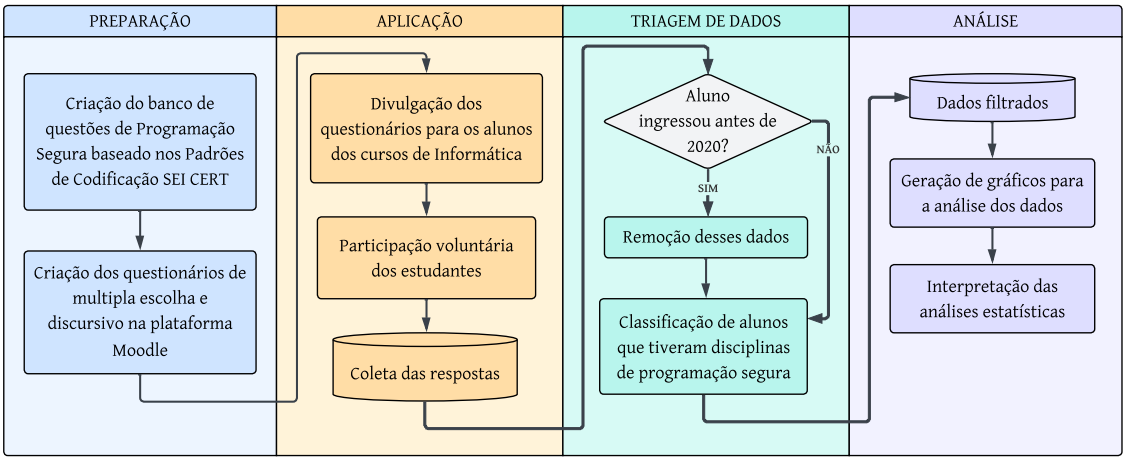

Figura 1. Etapas para a Realização da Pesquisa

Dois tipos de questionários foram disponibilizados aos alunos: *i)* questões de múltipla escolha, e *ii)* questões discursivas. Em ambos os casos, 5 questões de autoavaliação foram apresentadas no início do questionário aos participantes a fim de caracterizá-los. No questionário de múltipla escolha, existiam 10 questões técnicas (totalizando 15, com as de autoavaliação). Já no questionário com questões discursivas, existiam 6 questões

discursivas além das questões de autoavaliação. Os alunos podiam optar entre preencher apenas o questionário de múltipla escolha, ou preencher ambos os questionários (alunos que preencheram apenas o questionário discursivo foram desconsiderados).

As questões técnicas foram elaboradas baseadas nos Padrões de Codificação SEI CERT [CERT/SEI 2024] e pertenciam a uma de três categorias possíveis: Vetores, Strings e gerenciamento de memória. O escopo do estudo é apenas em programação segura na linguagem de programação C, e por isso a sintaxe das questões focou nessa linguagem. A seguir, cada classe é brevemente apresentada, juntamente com um exemplo de questão.

Vetores – Questões que focam em verificar a capacidade dos alunos em identificar problemas relacionados ao uso de vetores, como indexação e alocação incorretas.

Exemplo de questão:

No código abaixo, qual é o problema que pode levar a um comportamento indefinido?

```
enum { TABLESIZE = 100 };
static int table[TABLESIZE];

int *f(int index) {
  if (index < TABLESIZE) {
      return table + index;
  }
  return NULL;
}
```

a. O código não verifica se o índice é maior que TABLESIZE.
b. O código não retorna um ponteiro válido.
c. O código não verifica se o índice é negativo.
d. O código não aloca memória suficiente para o vetor.

Strings – De forma similar aos vetores, as questões relativas a strings almejam verificar a capacidade dos alunos em identificar problemas como acesso à índices inválidos, *buffers overflows* e afins.

Exemplo de questão:

Por que o código abaixo é inseguro, mesmo compilando sem *warnings*?

```
char *filename = "config.txt";
filename[0] = 'C';  // Tenta mudar 'c' para 'C'
```

a. Falta usar malloc() para alocar filename.
b. O tipo char* não é compatível com strings literais.
c. O compilador não permite a reatribuição de filename.
d. String literais são armazenadas em memória somente leitura, e modificá-las causa comportamento indefinido.

Gerenciamento de Memória – Questões relativas ao gerenciamento adequado de memória, incluindo alocação dinâmica de variáveis e problemas relacionados ao escopo de variáveis.

Exemplo de questão:

Qual das seguintes afirmações é verdadeira sobre o uso de ponteiros para memória que já foi liberada?

a. Tanto ler quanto escrever na memória liberada é comportamento indefinido.
b. É seguro ler a memória após a liberação, mas não escrever nela.
c. O ponteiro automaticamente se torna inválido e não pode ser mais usado.
d. A memória liberada é automaticamente limpa pelo sistema operacional, garantindo que nenhum dado permaneça acessível.

Vale notar que o foco das questões técnicas era verificar a capacidade dos alunos em identificar problemas de programação que podem deixar softwares vulneráveis. Por exemplo, problemas de *buffer overflow* em vetores e strings são conhecidos por gerar tais brechas de segurança e tanto ponteiros quanto alocação dinâmica de memória são assuntos muitas vezes lecionados sem preocupação de associação com potenciais problemas de corrupção de memória, principalmente em turmas de programação introdutórias.

Ao todo, 85 alunos responderam apenas as questões de múltipla escolha, enquanto 39 alunos responderam tanto as questões de múltipla escolha quanto as questões discursivas. Na Figura 2 é exibida a quantidade de alunos que responderam aos questionários de acordo com o ano de ingresso. Alunos que ingressaram antes do ano de 2020 foram desconsiderados das análises devido ao baixo número de respondentes (6 alunos).

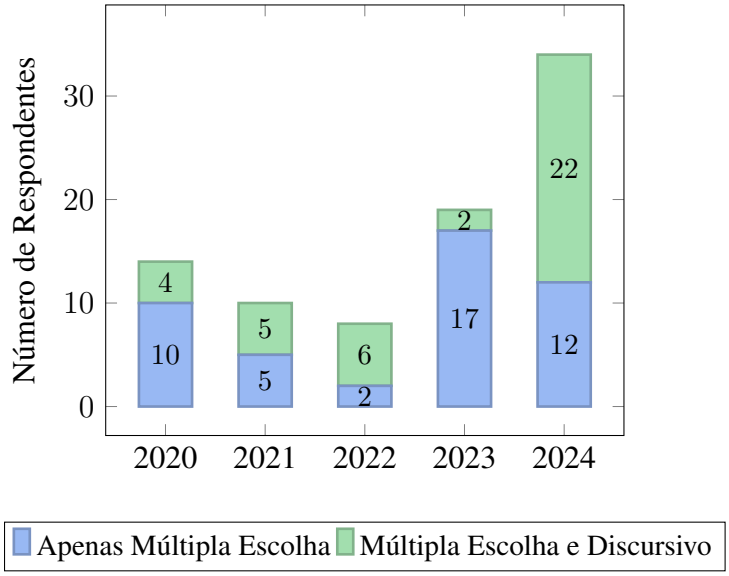

Figura 2. Distribuição dos respondentes por tipo de resposta e ano de ingresso

4. Resultados e Discussão

Esta seção apresenta e discute os resultado obtidos. Apresentamos os resultados das questões de múltipla escolha (Seção 4.1), seguido dos resultados das questões discursivas (Seção 4.2).

4.1. Questões de Múltipla Escolha

A Tabela 1 mostra os resultados médios obtidos pelos alunos de acordo com suas autoavaliações. Como pode ser observado, alunos que se julgaram como tendo algum conhecimento de programação segura obtiveram notas médias levemente superiores aos alunos

que se julgaram como não tendo conhecimento. Devido à baixa diferença, foi executado um Teste Mann-Whitney-Wilcoxon, que com nível de significância de $p = 0.95$ não rejeitou a hipótese nula de que as distribuições dos dois grupos de alunos eram as mesmas. Em outras palavras, de acordo com o teste, não é possível identificar diferença significativa entre os grupos.

	Julga que aprendeu conceitos de programação segura?	
	Sim, mesmo que parcialmente.	Não aprendeu ou não lembra.
# Alunos	58	27
Nota Média (desvio)	6.6 ($\pm$ 2.2)	6.3 ($\pm$ 2.1)

Tabela 1. Desempenho dos alunos de acordo com suas autoavaliações.

O resultado da Tabela 1 é interessante e preocupante, pois mostra um possível efeito Dunning-Kruger [Kruger and Dunning 1999], onde os estudantes podem ter lacunas tão grandes em seus conhecimentos que podem ser incapazes de identificar suas falhas, nesse caso, se julgando incorretamente como capazes de criar programas seguros.

Já na Tabela 2 são exibidos os resultados obtidos pelos alunos que assistiram ao menos uma disciplina em que pelo menos parte do conteúdo de programação foi ministrado incluindo conceitos de programação segura, versus os alunos que não tiveram. De forma similar aos resultados obtidos na Tabela 2, as notas obtidas são similares, e um Teste Mann-Whitney-Wilcoxon não identifica diferença significativa entre os grupos.

	Teve disciplinas com programação segura?	
	Sim, ao menos algumas.	Não.
# Alunos	64	21
Nota Média (desvio)	6.52 ($\pm$ 1.92)	6.62 ($\pm$ 2.78)

Tabela 2. Desempenho dos alunos que tiveram ou não disciplinas com programação segura.

Vale citar que são considerados alunos que tiveram disciplinas com conceitos de programação segura aqueles que, por exemplo, tiveram disciplinas introdutórias de programação, onde aprenderam a programar juntamente com alguns conceitos básicos de segurança como, por exemplo, validar se um ponteiro retornado pelo *malloc* é válido. O resultado mostra esse tipo de estratégia para o ensino de programação segura pode não ser suficiente [Lam et al. 2022].

Finalmente, na Figura 3 são exibidos os resultados médios dos alunos de acordo com seus respectivos anos de ingresso. A expectativa é de que alunos que entraram em anos anteriores, por terem passado mais semestres na universidade e serem mais experientes, sejam mais competentes em responder às questões de programação segura em C.

Os resultados apresentados na Figura 3 mostram alguns comportamentos interessantes. Primeiro, como esperado, os alunos que entraram mais recentemente nos cursos possuem uma grande variância nos seus resultados. A variância pode ser explicada pelo fato de alguns alunos já entrarem no curso com algum conhecimento de programação.

No entanto, de forma contraintuitiva, alunos que ingressaram no curso no ano de 2023 ou anteriormente não apresentaram notas médias significativamente maiores que os

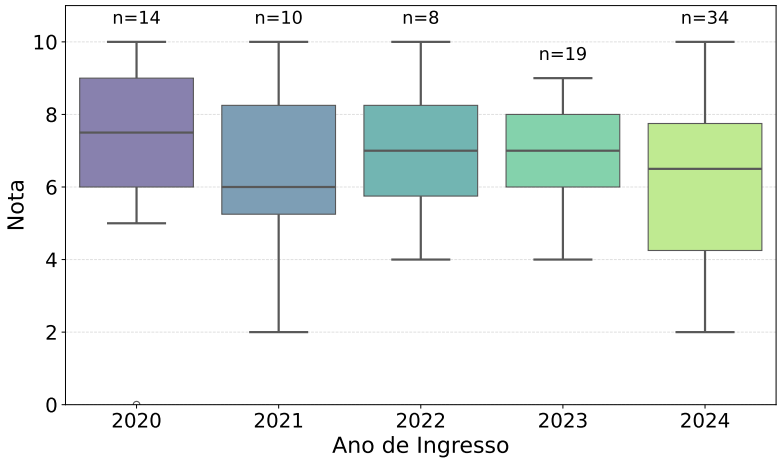

Figura 3. Distribuição das notas por ano de ingresso do aluno (questões de múltipla escolha)

alunos mais recentes, apesar de terem variâncias menores. Era de se esperar que alunos que cursaram mais disciplinas focadas em programação por estarem há mais tempo no curso melhorassem suas notas, mas esse fato não é observado nos dados. Mais interessante ainda são os resultados referentes ao ano de 2021, sendo esse o único ano em que os resultados obtidos pelos alunos foram piores que os dos recém ingressantes. Apesar de não estar no escopo deste trabalho, este pode ser um indicativo de que o período de pandemia da COVID-19, juntamente com a abordagem de aulas remotas, pode ter afetado negativamente o aprendizado dos alunos.

4.2. Questões Discursivas

Nesta Seção são apresentados os resultados considerando-se as questões discursivas. Como discutido na Seção 3, apenas 39 alunos responderam às questões discursivas. Os resultados das Tabelas 3 e 4 mostram, respectivamente, os resultados dos alunos de acordo com suas autoavaliações, e de acordo com a presença ou não de conteúdos de programação segura ministrados durante as disciplinas de programação.

	Julga que aprendeu conceitos de programação segura?	
	Sim, mesmo que parcialmente.	Não aprendeu ou não lembra.
# Alunos	23	16
Nota Média (desvio)	6.0 ($\pm$ 2.9)	5.2 ($\pm$ 2.1)

Tabela 3. Desempenho dos alunos de acordo com suas autoavaliações (questões discursivas)

Os resultados das tabelas levam às mesmas conclusões das análises realizadas na Seção 4.1, onde as diferenças entre os grupos de alunos foi relativamente pequena. Na Tabela 3 foi possível observar que alunos que julgam ter aprendido conceitos de programação segura tiveram um desempenho ligeiramente melhor quando comparado aos que não aprenderam ou não lembram, mas, ainda assim, essa diferença pode não ser significativa.

	Teve disciplinas com programação segura?	
	Sim, ao menos algumas.	Não.
# Alunos	30	9
Nota Média (desvio)	5.5 ($\pm$ 2.6)	6.3 ($\pm$ 2.5)

Tabela 4. Desempenho dos alunos que tiveram ou não disciplinas com programação segura (questões discursivas)

Um resultado particularmente intrigante emerge da Tabela 4, a qual evidencia que os alunos que tiveram aulas juntamente com conceitos básicos de programação segura obtiveram em média notas mais baixas quando comparados aos que não passaram por essa formação. Porém, mais uma vez, os grupos não possuem diferenças significativas e as diferenças nas notas provavelmente vieram do acaso. Além disso, devemos tomar esses resultados com cuidado devido à baixa amostragem de alunos que não tiveram programação segura na Tabela 4.

4.3. Discussão

Os resultados mostram que o ensino de técnicas de programação segura, juntamente com o ensino de técnicas de programação básica, como é o caso dos alunos que participaram desta pesquisa, não é efetivo. Considerando as perguntas de pesquisa propostas no início deste trabalho, obtemos as seguintes análises.

QP1: Há diferenças entre alunos que se consideram capazes de programar de forma segura e aqueles que não se consideram?

R.: Não há diferença significativa nos resultados obtidos pelos alunos que julgavam conhecer, ao menos em partes, programação segura, versus os demais. Essa é uma das principais descobertas deste trabalho, onde concluímos que os estudantes estão criando uma falsa percepção de que têm conhecimentos de programação segura, podendo os levar a erros crassos devido ao excesso de confiança.

QP2: Os alunos que tiveram conceitos de programação segura enquanto aprendiam a programar, programam de maneira mais segura do que os demais?

R.: Mais uma vez, não há diferença significativa entre os grupos. Essa descoberta mostra que as formas de ensino de programação segura nos cursos são insuficientes ou ineficazes para o aprendizado dos alunos.

QP3: Os alunos melhoram a segurança de suas programações conforme progridem no curso, ou existe um plateau em que os alunos deixam de progredir?

R.: Os alunos melhoram a qualidade de suas programações (com relação à segurança) entre seus primeiro e segundo anos nos cursos. No entanto, os alunos atingem um plateau de conhecimento já no segundo ano nos cursos. Isso mostra, mais uma vez, que o ensino de técnicas de programação segura não estão sendo suficientes nos cursos, e isso persiste durante os vários semestres letivos que os alunos cursam.

Para mitigar esses problemas, recomenda-se que a criação de disciplinas focadas apenas em programação segura, na qual alunos já com alguma experiência em programação podem se matricular. Dessa forma, os alunos podem focar apenas nos conceitos de segurança, sem a carga cognitiva extra de, em paralelo, precisarem aprender a progra-

mar. Tal disciplina existe nos cursos analisados, no entanto, essa disciplina se encontra no final do curso, e a vasta maioria dos alunos que responderam aos questionários não a cursaram. Além disso, tal disciplina pode focar, por exemplo, na aprendizagem focada em problemas, a fim de aumentar o engajamento dos alunos.

5. Conclusão e Trabalhos Futuros

Os resultados deste trabalho mostram que os alunos que tiveram conteúdos de programação segura em conjunto com conteúdos básicos de programação não foram capazes de absorver os conteúdos corretamente, não sendo melhores do que seus pares que não tiveram tais conteúdos de programação segura. Além disso, o fato de grupos de alunos que se consideram capazes de programar seguramente não ser capaz de superar o grupo que não se considera capaz de programar seguramente mostra que os alunos podem estar criando uma falsa percepção sobre suas capacidades de criar softwares seguros.

Apesar de a literatura de codificação segura ter aproximadamente 25 anos, ainda se vê a necessidade de integrar a programação segura nos currículos de computação, apresentando estratégias pedagógicas baseadas em referências consolidadas, como os padrões do CERT e obras clássicas de segurança de software. Sem essa mudança curricular e sem o incentivo a se ter disciplinas específicas de segurança em todo o ciclo de desenvolvimento de software, os cursos relacionados à computação continuarão a formar desenvolvedores que, mesmo competentes tecnicamente, perpetuam vulnerabilidades evitáveis.

Vale notar alguns aspectos e limitações desta pesquisa, dentre eles: 1 – A grande maioria dos alunos pesquisados não cursaram disciplinas focadas apenas em programação segura, pois não há oferta no currículo, dessa forma, a análise se limita aos alunos que cursaram apenas disciplinas de programação básica onde alguns conceitos de programação segura são introduzidos em assuntos como alocação dinâmica de memória. 2 – A análise foca em alunos de uma única instituição e; 3 – O fato de que poucos alunos responderam às questões dissertativas pode ter gerado limitações nas análises.

Contudo, o presente trabalho mostra a importância de pesquisas sobre as habilidades de programação segura de alunos em cursos de computação em geral, a fim de se identificar problemas e limitações em sua formação. As descobertas deste trabalho servirão para discussões sobre a reorganização da grade curricular dos cursos de computação a fim de, por exemplo, trazer mais conteúdos focados primariamente em programação segura e o ciclo de desenvolvimento seguro de software, pensado para se adicionar segurança desde o projeto de um software, bem como o de reorganizar disciplinas focadas em segurança na matriz curricular a fim de que estas sejam absorvidas da melhor forma possível pelos estudantes. O objetivo principal de se dar ênfase para disciplinas específicas em segurança é o de formar desenvolvedores capazes de revisar criticamente códigos e evitar vulnerabilidades comuns.

Como trabalho futuro, pretende-se expandir a pesquisa para coletar mais dados com participantes sobre programação segura tanto em C quanto em outras linguagens de programação, na Universidade Federal do Paraná e em outras instituições colaboradoras do Brasil ou do exterior, considerando-se também alunos de pós-graduação.

Referências

Barnum, S. and McGraw, G. (2005). Knowledge for software security. *IEEE Security and Privacy*, 3(2):74–78.

CERT/SEI (2024). CERT secure coding standards. `https://wiki.sei.cmu.edu/confluence/display/seccode`.

Chi, H., Jones, E. L., and Brown, J. (2013). Teaching secure coding practices to stem students. In *Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference*, InfoSecCD '13, page 42–48, New York, NY, USA. Association for Computing Machinery.

Espinha Gasiba, T., Oguzhan, K., Kessba, I., Lechner, U., and Pinto-Albuquerque, M. (2023). I'm Sorry Dave, I'm Afraid I Can't Fix Your Code: On ChatGPT, CyberSecurity, and Secure Coding. In Peixoto de Queirós, R. A. and Teixeira Pinto, M. P., editors, *4th International Computer Programming Education Conference (ICPEC 2023)*, volume 112 of *Open Access Series in Informatics (OASIcs)*, pages 2:1–2:12, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Evans, D. and Larochelle, D. (2002). Splint - annotation-assisted static program checker. `https://splint.org/`.

Hong, H., Woo, S., and Lee, H. (2021). Dicos: Discovering insecure code snippets from stack overflow posts by leveraging user discussions. In *Proceedings of the 37th Annual Computer Security Applications Conference*, ACSAC '21, page 194–206, New York, NY, USA. Association for Computing Machinery.

Howard, M. and LeBlanc, D. (2003). *Writing Secure Code*. Microsoft Press, 2 edition.

Kruger, J. and Dunning, D. (1999). Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of personality and social psychology*, 77(6):1121.

Lam, J., Fang, E., Almansoori, M., Chatterjee, R., and Soosai Raj, A. G. (2022). Identifying gaps in the secure programming knowledge and skills of students. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*, SIGCSE 2022, page 703–709, New York, NY, USA. Association for Computing Machinery.

McGraw, G. (2006). *Software Security: Building Security In*. Addison-Wesley.

Pugh, B. and Loskutov, B. (2015). Findbugs - find bugs in java programs. `https://findbugs.sourceforge.net/`.

Rahman, A., Farhana, E., and Imtiaz, N. (2019). Snakes in paradise?: Insecure python-related coding practices in stack overflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 200–204.

Seacord, R. C. (2013). *Secure Coding in C and C++*. Addison-Wesley, 2 edition.

Singleton, L., Zhao, R., Song, M., and Siy, H. (2020). Cryptotutor: Teaching secure coding practices through misuse pattern detection. In *Proceedings of the 21st Annual Conference on Information Technology Education*, SIGITE '20, page 403–408, New York, NY, USA. Association for Computing Machinery.

Taylor, B., Bishop, M., Hawthorne, E., and Nance, K. (2013). Teaching secure coding: the myths and the realities. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, page 281–282, New York, NY, USA. Association for Computing Machinery.

Viega, J. and McGraw, G. (2001). *Building Secure Software*. Addison-Wesley.

Votipka, D., Fulton, K. R., Parker, J., Hou, M., Mazurek, M. L., and Hicks, M. (2020). Understanding security mistakes developers make: qualitative analysis from build it, break it, fix it. In *Proceedings of the 29th USENIX Conference on Security Symposium*, USA. USENIX Association.