Editors: Melissa Dark, dark@purdue.edu | Jelena Mirkovic, mirkovic@isi.edu | Bill Newhouse, william.newhouse@nist.gov

Hello, World!—Code Responsibly

Siddharth Kaza, Blair Taylor, and Kyle Sherbert | Towson University

The digital landscape has changed dramatically since the first "hello, world" program. With software pervasive to every facet of our lives, writing secure code has never been more important. Many security exploits target vulnerabilities that are a result of poorly designed and implemented software. Despite its critical importance, secure coding is primarily taught in upper-level elective computer science (CS) classes. As a result, most students graduate with little or no exposure to secure coding techniques. We believe that it is important to teach responsible coding early, starting in the first programming course, and often, by repeating and reinforcing security concepts in advanced courses.

Towards this end, some schools are working to incorporate secure coding into the CS curriculum. Researchers at Stanford University and Purdue University have developed a game for teaching secure coding;[1] faculty at the University of North Carolina, Charlotte (UNCC) have built an interactive integrated development environment (IDE) to detect vulnerabilities;[2] and the University of California, Davis, has teamed up with researchers at Purdue to create a secure programming clinic to assist programming students in writing robust programs.[3] At Towson University, we have developed modules that teach students to code securely and responsibly from the first class.

The Security Injections @Towson Project

Started in 2006, the Security Injections @Towson project includes over 50 learning objects (Table 1). The Security Injection nano-modules target important security concepts that can be addressed effectively in introductory CS courses (CS0, CS1, CS2), including integer error, buffer overflow, input validation, and data hiding. Individual modules were developed for languages commonly used in these courses—C++, Java, and Python. Each module requires minimal instructor intervention, is auto-graded, and has been shown to be effective in both classroom and online courses.[4–6]

Due to the nature of the field, many CS and cybersecurity labs are developed rapidly in an ad hoc fashion and lead to a "run-and-submit" mentality in students without encouraging reflection. The Security Injection nano-modules (Figure 1) follow a consistent format grounded in the learning sciences and include the following sections:

- Background—This section provides context for the security vulnerability, activates background knowledge, and stimulates learning.
- Code Responsibly—This section contains strategies that programmers should employ to avoid these vulnerabilities. At the end of both this section and the Background, we pose multiple-choice comprehension questions to encourage students to thoroughly read the material and to think about what they have just learned.
- Lab Assignment—In this section, students write programs that demonstrate the vulnerabilities and apply the strategies from the previous section.
- Security Checklist—This section encourages students to analyze blocks of code, allowing them to interactively trace the source of vulnerabilities and highlight potential areas of concern (see Figure 2). Our project's approach is to reinforce concepts at increasing levels of learning; for instance, the checklists teach students to identify buffer overflows in CS1 and techniques for mitigation in CS2. In addition, repeated exposure to the checklists or code reviews allows students to become skilled in self-checking their code.
- Discussion—This section includes questions that require further research and reflection. The questions are open-ended, and their answers are recorded on a completion certificate.

It Takes a Village

The Security Injections project has been collaborative since the start, including faculty from community colleges and an HBCU (Historically Black College or University) to design, pilot, develop, and assess the learning objects. This allowed a continuous improvement process based on faculty feedback. Involving faculty throughout the process also increased instructor buy-in and adoption of materials.

Building effective materials is not enough; outreach is crucial to sustainability. We conducted over 50 faculty workshops, panels and birds-of-a-feather groups, a Build-A-Lab program, and a Security Ambassador program reaching over 400 faculty across 221 institutions, including more than 90 community colleges and several high schools (Figure 3). The goal

Copublished by the IEEE Computer and Reliability Societies

of each workshop was to increase faculty interest in our project and encourage them to include the Security Injections in their classes. The agenda included open discussions and breakout groups to promote feedback, reflection, and sharing of teaching strategies. Our workshops are designed to be as modular as our Security Injections: flexible enough to fit into various time allotments and comprehensive enough to be easily shared. We have a growing base of "security ambassadors" who conduct workshops on Security Injections at conferences and at local institutions across the country.

Table 1. Security Injections @Towson—learning object taxonomy.

Learning object	Completion time
Nano-module	Less than 1 hour
Micro-module	1 to 4 hours
Module	Between 4 hours and two weeks (6 hours)
Unit	Between 2 weeks and 15 weeks (45 hours)
Course	15 weeks (45 hours)

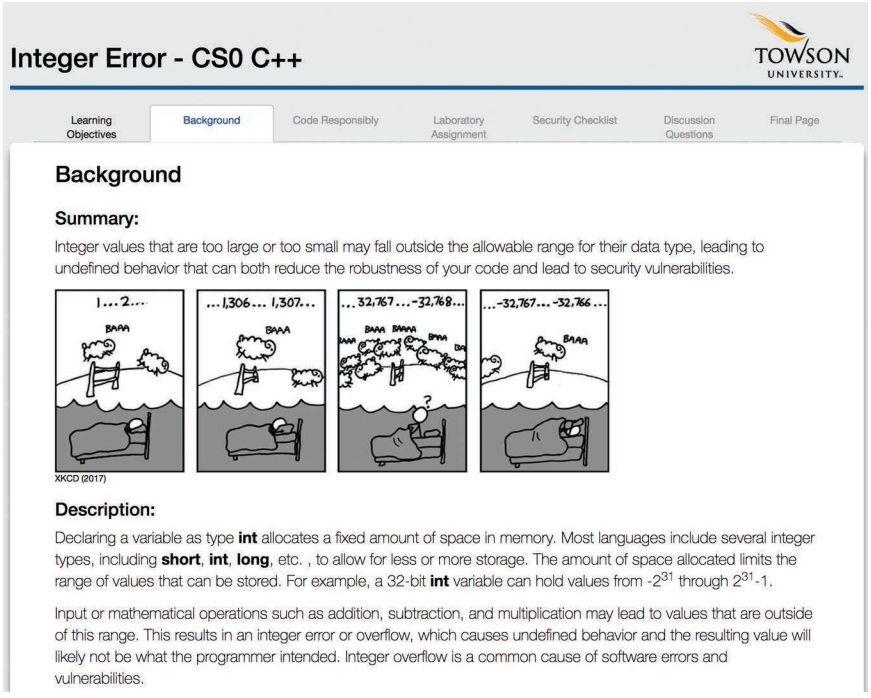


Figure 1. The Integer-error nano-module in Java for CS0.

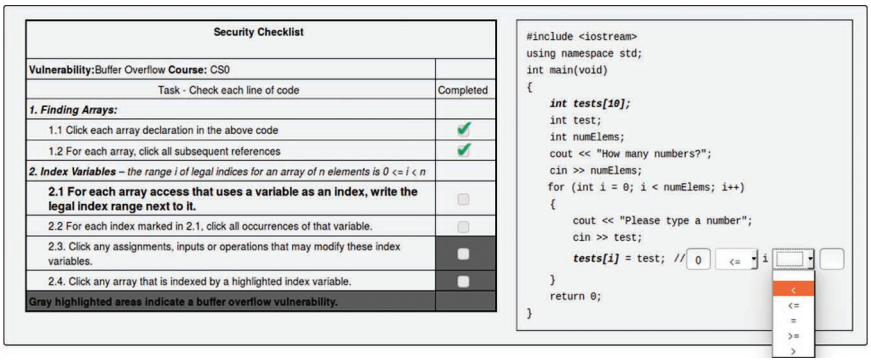
Tips to Introduce Responsible Coding in Your First Course

- Emphasize the importance of security in the software development lifecycle. Instead of using the traditional "hello, world!," try the following as your first programming lab:

```
print ('The Software
Development Lifecycle')
print ('1: Analysis: Define
the problem')
print ('2: Design: Plan the
solution')
print ('3: Implement: Code
the solution')
print ('4: Test and debug')
print ('5: Maintain and
document')
print ('6: Add security
features')
```

As a follow-up, students must remove step 6 and integrate security into each step. It's a very simple exercise—not much harder than printing "hello world"—that introduces responsible coding.

- Include secure programming as a course objective in your syllabus. Including objectives and learning outcomes like those in the ACM/IEEECS2013(https://www.acm.org/education/curricula-recommendations), NSA/DHS Center of Academic Excellence (CAE) knowledge units (https://www.nsa.gov/resources/educators/centers-academic-excellence/cyber-defense), and Joint Task Force on Cybersecurity Education guidelines (https://www.csec2017.org) will facilitate accreditation and designation processes. To reiterate the importance of secure coding, include the Code Responsibly icon (Figure 4), or a similar mantra, on the syllabus.
- Capitalize on students' interest in security. Our experience indicates that most students, including females and minorities, have high interest in cybersecurity. This inherent interest in security can be used to motivate students to learn, for example, teaching binary in the context of integer errors. Think outside the box for your



Security Checklist	
Vulnerability: Buffer Overflow **Course:** CS0	
Task - Check each line of code	Completed
1. Finding Arrays:	
1.1 Click each array declaration in the above code	✔
1.2 For each array, click all subsequent references	✔
2. Index Variables – the range i of legal indices for an array of n elements is 0 <= i < n	
2.1 For each array access that uses a variable as an index, write the legal index range next to it.	
2.2 For each index marked in 2.1, click all occurrences of that variable.	
2.3. Click any assignments, inputs or operations that may modify these index variables.	
2.4. Click any array that is indexed by a highlighted index variable.	
Gray highlighted areas indicate a buffer overflow vulnerability.	

```
#include <iostream>
using namespace std;
int main(void)
{
    int tests[10];
    int test;
    int numElems;
    cout << "How many numbers?";
    cin >> numElems;
    for (int i = 0; i < numElems; i++)
    {
        cout << "Please type a number";
        cin >> test;
        tests[i] = test; // 0 <= i
    }
    return 0;
}
```

Figure 2. The buffer overflow Security Checklist in CS0—C++. Students begin by clicking each vulnerable array wherever it appears; they must then specify the legal index range for that array, and trace changes to any variable used as its index.

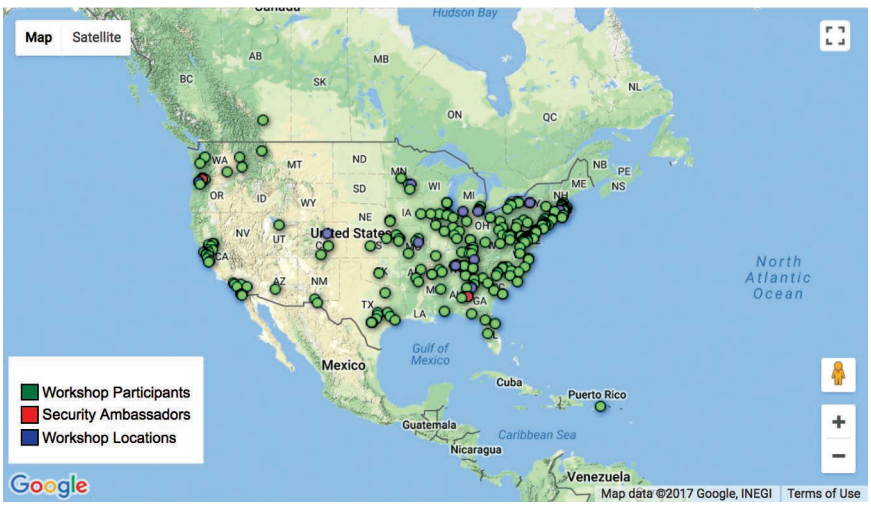


Figure 3. Security Injections workshop participants.

Figure 4. The Code Responsibly button.

assignments—a creative project can effectively demonstrate student knowledge. Below is the final stanza from a poem written by a CS0 student:

When I fixed my program, a thought came to me,

Input Validation is all about security,

If used correctly it makes my programs robust,

So I can be a programmer you trust.

The Security Injections learning objects are interactive, auto-graded, and freely available at www.towson.edu/securityinjections.

Code responsibly! ■

Acknowledgments

The Security Injections @Towson and its ancillary projects are supported by the National Science Foundation under grants NSF DUE-1241738, NSF DUE-0817267, NSF DGE-1516113, NSF DGE-1516113, NSF DGE-1241649, the National Security Agency, and the Intel Corporation.

References

1. N. Adamo-Villani, M. Oania, and S. Cooper, "Using a Serious Game Approach to Teach Secure Coding in Introductory Programming: Development and Initial Findings," *Journal of Educational Technology Systems*, vol. 41, no. 2, 2012–2013, pp. 107–131.
2. J. Xie, B. Chu, and H.R. Lipford, "Idea: Interactive Support for Secure Software Development," *Proceedings of the Third International Conference on Engineering Secure Software and Systems* (ESSoS 11), 2001, pp. 248–255.
3. I. Ngambeki et al., "Teach the Hands, Train the Mind … A Secure Programming Clinic!," *Proceedings of the 19th Colloquium for Information Systems Security Education*, 2015, pp. 119–133.
4. B. Taylor and S. Kaza, "Security Injections: Modules to Help Students Remember, Understand, and Apply Secure Coding Techniques," *Proceedings of the 16th Annual Conference on Innovation and Technology in Computer Science Education* (ITICSE 11), 2011.
5. B. Taylor and S. Kaza, "Security Injections @Towson: Integrating Secure Coding into Introductory Computer Science Courses," *ACM Transactions on Computing Education*, vol. 16, no. 4, 2016.
6. S. Raina, S. Kaza, and B. Taylor, "Security Injections 2.0: Increasing Ability to Apply Secure Coding Knowledge Using Segmented and Interactive Modules in CS0," SIGSCE, 2016, pp. 144–149.

Siddharth Kaza is an associate professor at Towson University. Contact at skaza@towson.edu.

Blair Taylor is a clinical associate professor at Towson University. Contact at btaylor@towson.edu.

Kyle Sherbert is a graduate student at Towson University. Contact at ksherb1@students.towson.edu.