Teaching High School Students to Code Responsibly

Kara Nance, University of Alaska Fairbanks and Blair Taylor, Towson University

Abstract –Teaching secure programming should not be separate from teaching programming. By teaching high school students to code responsibly we foster a security mindset and establish a foundation of secure programming skills. High school computing teachers need ready-to-use resources that allow them to incorporate security principles in their programming classes. Additionally, it would be helpful to provide a seamless laboratory environment in which to run them. This paper discusses the Security Injections @ Towson project running in the RAVE environment which has proven success in the two- and four- year environment and could easily be adapted for the high school classroom.

Index terms: Security Injections, Secure Programming, Lab Modules, Security Integration, Education

I. Introduction

Experts agree that one of the key factors in cybersecurity education is creating a security mindset [1]. With this in mind, it is critical that security education begin early and continue throughout the students' educational careers. This is particularly true for secure programming. When students learn programming, they should learn how to program securely and this should start with early programming courses and be reinforced throughout subsequent courses.

As colleges strive to integrate security and secure coding into their curriculum, a similar effort is necessary at the high school level. Many students first learn to program in high school. Learning secure principles at this level would go a long way towards establishing responsible coding in the future. It also means less undoing of insecure coding practices down the road. Some of the identified challenges in security education include the following:

1) A lack of a security culture or an infrastructure/administration that does not recognize the importance of security
2) Inadequate security teaching skills coupled with a lack of confidence
3) Challenging laboratory environments

These challenges can be even greater for high school teachers, many of whom have little formal computer training and suffer from a woeful lack of resources and support. High school computing teachers need resources that allow them to understand critical secure coding principles that are not addressed in most textbooks. Additionally, they need ready-to-use materials they can *incorporate into their programming classes. Ideally, it would be helpful to have a seamless laboratory environment in which to run them. Finally, it is critical that high school teachers and administrators understand the importance of security.*

For the past several years, we have been implementing shareable security modules across the college curriculum and assessing the results. Results indicate an increase in students' ability to understand and apply security principles [2]. This paper describes the design and implementation of the security injections and instructions for implementation at the high school level.

II. Related Work

Security is a relatively new challenge in the dynamic field of computer science. With national pressure to increase cybersecurity, there has been increased attention in both industry and education towards secure software and systems development [3]. Since most security threats are the result of system or software vulnerabilities, it is imperative that all current and future software developers are proficient in secure design and programming. The importance of integrating or threading security across the computing curriculum, beginning with the foundation courses and re-enforcing throughout upper level courses is widely recognized [1]. But in reality, work at the college level has just begun and integrating security into high school courses is an even newer challenge.

III. Previous Results

The Security Injections @ Towson project (www.towson.edu/securityinjections) began in response to a pressing need to increase the number of security skilled IT professionals. The first step in this response for many universities was to offer security courses and security tracks, but these efforts were only reaching limited number of students. The goal of the Security Injections @ Towson project was to develop a breadth-first security integration program, which included the creation of security injection modules, a detailed assessment strategy, and faculty training and outreach. To date, the project includes over 45 modules, including modules for the introductory computing course for non-

majors and secure coding modules for the introductory programming courses. The secure coding modules for CS1 and CS2 are available in Java and C++ and also available in Python and pseudocode for CS0. The modules have been tested at the community college and university levels. The project has been disseminated nationally and internationally to numerous faculty and students and we have partnered with other educational efforts to ensure that these free resources are utilized to help prepare an ever-expanding cadre of technology users and educators.

In 2010, Towson University partnered with the University of Alaska Fairbanks to bring the security injections to an expanded audience via the NSF-funded Remote Accessible Virtualized Environments (RAVE) Project. The goals of both projects are:

a) To integrate security throughout the curriculum
b) Lay the foundation towards creating a security mindset at the earliest courses
c) To reiterate security concepts early and often
d) To facilitate security education by creating a library of security materials [4]

The RAVE project was developed in response to an urgent need for a platform to facilitate the educational use of technology laboratories. While the range of capabilities available in the RAVE are extensive, the most important aspect that they bring to the table when combined with the Security Injections @ Towson is the ability for the security injects to be executed without requiring the instructor to set up special resources to observe the behavior of the inject as demonstrated in the following sections which describe the module topics, the format, and provide some examples.

IV. Overview

Security Injections. A security injection module is a laboratory module that covers a key security concept. The structure of each module, constructed as a scientific laboratory exercise, is to foster active learning and synthesis of important security principles. Each module is designed to be completed independently by the student and included as a standalone assignment or as a part of a larger assignment. Each security injection module includes the background information, a laboratory exercise hosted on the RAVE environment, a security checklist, and discussion questions pertaining to a particular security issue. Most modules take less than 30 minutes to complete.

A. Module Topics

Programming: Each security injection module targets a specific security topic. For the purpose of our project, we identified topics by researching industry needs assessment lists such as CVE and SANS Top Ten Vulnerabilities [5,6]. While there are many options for modules that would fit in the high school curriculum, selected starting modules include 1) integer error, 2) input validation, and 3) buffer overflow. Including these topics in introductory programming courses will go a long way towards reducing some of the most prevalent vulnerabilities in application and system software.

In addition to the topics that could be easily coupled with CS1 and CS2 content, there are computer literacy modules that address general security topics, including Password, Cryptography, and Phishing. These modules are suitable for general IT courses or can be used to supplement the programming courses.

B. Module Format

Each security module has undergone pilot testing and numerous revisions to yield an easy to use and student-friendly format. This format is adaptable to many levels of education. Each security injection module includes the following sections:

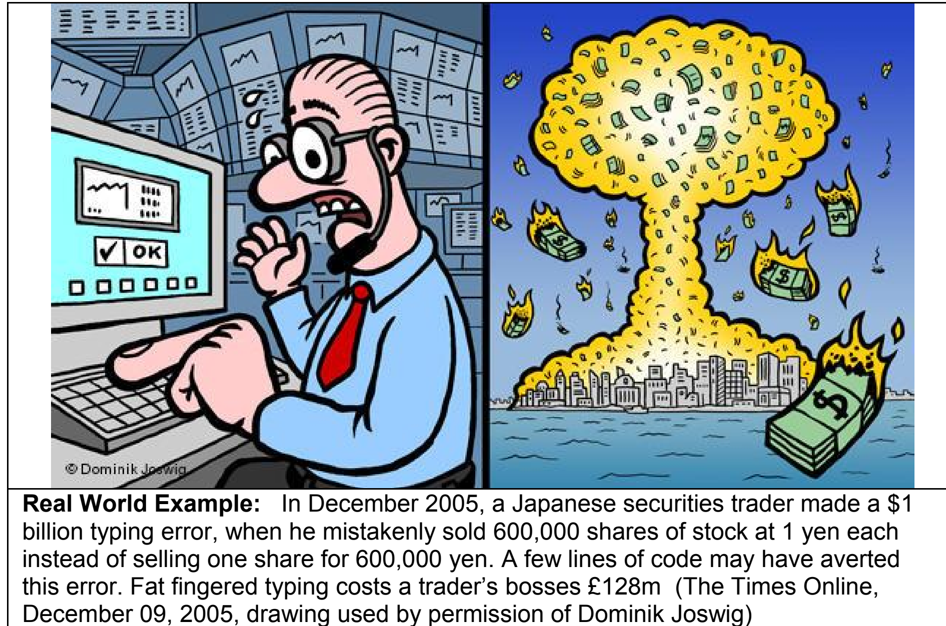
1) Background
2) Laboratory exercise
3) Security Checklist
4) Discussion questions.

More detail with examples for each section is provided below. Additionally, each module begins with a catchy or clever title to help catch the student's attention and create a visual image. Example titles include:

Passwords: "Long, Strong, but Memorable"
Cryptography – "Scrambling information"
Integer Error –"You can't count that high"
Input Validation – "All Input is Evil"
Buffer Overflow - "Data gone wild"

1. Background

The first component of the security inject includes a brief description of the security issue. The *Background* section also includes at least one real-life occurrence. Most of the modules include a comic strip or graphic to help student visualize the concepts. For example, the cartoon in figure 1, a man with a "fat finger" issue is depicted to illustrate the input validation problem and reinforces the accompanying real world example.



Real World Example: In December 2005, a Japanese securities trader made a $1 billion typing error, when he mistakenly sold 600,000 shares of stock at 1 yen each instead of selling one share for 600,000 yen. A few lines of code may have averted this error. Fat fingered typing costs a trader's bosses £128m (The Times Online, December 09, 2005, drawing used by permission of Dominik Joswig)

Figure 1 Graphic and Real-World Example

2. Code Responsibly

There are many aspects to programming correctly, including programming for efficiency, user-friendly programming, and programming defensively for safety and security, also known as robust programming. To coin a phrase that students can easily identify with, we use the term *code responsibly* to include all of the above. To reinforce the concept, we created buttons and include the icon as seen in figure 2 on our course materials.

Each programming security injection modules includes a *Code Responsibly* section which includes tips to avoid the targeted error. The Code Responsibly tips will be particularly appropriate for high school programming courses, such as the AP Computer Science. Figure 2 shows the Code Responsibly section which is associated with a module on Input Validation.

Code Responsibly – How can I properly validate input?

Check all input: Below is a partial list of some checks to include:

- **Range check** - numbers checked to ensure they are within a range of possible values, e.g., the value for month should lie between 1 and 12.
- **Reasonableness check**: values are checked for their reasonableness, e.g. (age > 16) && (age < 100)
- **Divide by Zero**: variables are checked for values that might cause problems such as division by zero.
- **Format check**: Checks that the data is in a specified format (template), e.g., dates have to be in the format DD/MM/YYYY.

Figure 2 Code Responsibly Example for Input Validation

3. Laboratory Exercises

The second part of the security module is a brief hands-on lab exercise that gives the students practice with the concept. Figure 3 provides an example of an input validation lab exercise for an introductory programming course, such as AP Computer Science:

```
Program 1
import java.util.*;
public class InputExa
{
  public static void main(String[] args)
  {
    Scanner console = new Scanner(System.in);
    int age;
    int total = 0;
    float avgAge;

    System.out.println("Enter 10 ages: ");
    for (int i = 0;i < 10; i++)
    {
      age = console.nextInt();
      total = total + age;
    }
    avgAge = (float)total/10;
    System.out.println("average age is " + age);
  }
}
```

Figure 3 Java Input Validation Lab

This is the portion of the security injection that can benefit from use of the RAVE environment. Rather than having students locate an execution environment, locate a compiler and editor, type in the program, debug the program, and run the program, there is the option for the instructor to use the RAVE environment to complete the exercise. In addition, there is work being done to record the execution in the RAVE environment for viewing in a group setting or individually. Security injection lab

exercises vary greatly in scope and depth, depending on the course and level of students. Other laboratory exercises include evaluating an email for validity, running checks on a password, or entering, compiling, and running a program.

4. Lab Questions

Lab Questions:

1. Type* Program 1 and Compile. Run.
2. Complete the security checklist for this program. Submit marked program and completed checklist.
3. What variable(s) should be validated and describe what you should check for?
4. Could integer overflow occur for the variable total? How?
5. Optional: Change the program to properly validate the input.

Figure 4 Lab Questions for Input Validation Module

Each laboratory exercise has associated instructions, questions which guide the student through the lab, and a security checklist. The instructions and associated lab questions for the input validation module are shown in figure 4.

5. Security Checklist

Security Checklist	
Vulnerability: *Failure to Validate Input* **Course:** *CS1*	
Check each line of code	
1. Mark with a V each variable that is input.	
For each V, which of the following is applicable	
1. Check length?	
2. Check range (reasonableness)?	
3. Check format?	
4. Check type?	
Highlighted areas indicate vulnerabilities!	

Figure 5 Security Checklist Input Validation

Checklists have many applications, including software assurance. Each of the security injection modules includes a security checklist. The use of security checklists serve several purposes: it teaches students to manually find crucial vulnerabilities in their code; it reinforces important security principles through self-evaluation; and the use of checklists initiates good code-checking habits. Completing the security checklists is a form of self-evaluation that allows students to internalize important security topics [7]. An example security checklist associated with an Input Validation Module, shown in figure 5, forces students to think about what type of input checking they should include in their programs.

6. Discussion Questions

The final component of the security injection module are discussion questions that include feedback questions that ask students to reflect back on their work and further synthesize the security concept and research questions that require the student to look deeper into the topic. Figure 6 shows the sample discussion questions for an input validation security injection module for a programming course.

Discussion Questions

1. Can improper input validation lead to integer errors?
2. Explain how range and reasonableness checking can help companies prevent errors.
3. Why is a while loop usually a more effective way to perform input validation than an if..else?
4. Accepting known good values is known as whitelisting. Rejecting bad values is known as blacklisting. Write the loop construct for whitelisting a body temperature. Write the loop construct for blacklisting a body temperature. Why is whitelisting much stronger?
5. What is a “fat finger”? List ways to make you program “fat finger friendly.”

Further Work (optional – check with your instructor if you need to answer the following questions)

1. Filenames are particularly vulnerable to security vulnerabilities. Research to find out why.
2. Another important security strategy is “defense in depth”. Explain what you think this means. How could this relate to input validation?

Figure 6 Discussion Questions for Input Validation

C. Administration Details

For an introductory high school programming course, security injections can be included as part of existing assignments or added as extra labs. An example of an introductory programming course at Towson that includes

the injections as supplementary labs by including links to each injection on the course syllabus can be found at: (http://pages.towson.edu/btaylor/cosc175/syllabus/syll_175.htm), under Course Schedule and Assignment. In this course, the security injections are assigned as follows:

Course Topic	Security injection
Data and Variables	*Integer Error*
Loops	*Input Validation*
Arrays	*Buffer Overflow*

Students are expected to submit a copy of their program, sample output, completed checklists and marked up code, as well as answers to lab and discussion questions. Answer keys for assignments are available upon request. The security injection modules are intended to be stand-alone with minimal help required from the teacher. Feedback has been positive from faculty that has used the modules, and some reported that the security injections helped them feel more comfortable with security topics.

V. DISCUSSION

While the security injection exercises have been well-received and widely used in higher education, programming courses and technology courses are offered at all levels of education. Increasing user awareness of security issues and teaching secure programming or responsible programming at all levels is increasingly important. Many of the existing security injections can be easily adapted (and are of interest to) K-12 populations.

Dissemination and adoption remain significant challenges and *systemic curriculum change requires broader commitment. Computing teachers at all levels are beginning to recognize the need for security, but lack requisite security knowledge and resources, and are constrained by time. As with most cultural changes, we cannot expect an instant conversion, even if most parties agree that the change is in the best interest of the participants. We can help stimulate this change by providing the valuable resources that high school teachers need in order to effectively present the materials to their student audiences easily and without a significant duplication of effort.*

Acknowledgments: This research was supported in part by NSF CCLI Grant DUE-0817267and NSF CCLI Grant DUE-1023125.

VI. REFERENCES

[1] Burley, D. & M. Bishop. Final Report. Summit on Education in Secure Software. June 1, 2011.

[2] Taylor, B. & Kaza, S. (2011). Security Injections: Modules to Help Students Remember, Understand, and Apply Secure Coding Techniques, ITICSE 11

[3] Westervelt, R. Educators see secure coding training challenges, improvements. Search Security.com http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1346086,00.html, Accessed February 12, 2009.

[4] Nance, K. Taylor, B., Dodge, R. & Hay, B. (2011) Creating Shareable Security Modules, WISE Proceedings

[5] Christey, S, 2009 CWE/SANS Top 25 MostDangerous Programming Errors, http://cwe.mitre.org/top25/, Accessed 13 February 2009.

[6] SANS Institute, New Report Identifies the Three Programming Errors Most Frequently Responsible For Critical Security Vulnerabilities and Security Incidents in 2006, retrieved February 12, 2009 from http://www.ssi-sans.org/resources/top_three.pdf.

[7] Taylor, B. and S. Azadegan, "Using Security Checklists and Scorecards in CS Curriculum," National Colloquium for Information Systems Security Education, 2007, pp. 4-9