Teaching Secure Coding Practice to Novice Programmers : Creating CS1/2 Laboratory Modules in the Context of Security

Su Hyun Kim, Steve Cooper(Mentor), Nicoletta Adamo-Villani(Purdue)

Contact: suhyunk@stanford.edu

Special thanks to Professor Steve Cooper and John-Ashton Allen

Introduction

•Most undergraduate CS curricula do not include secure coding practices.

•Textbooks on secure coding are targeted at the advanced undergraduate or graduate level.

•Rather than "un-teaching" bad coding habits later in their education, a more effective measure would be to teach secure programming practices to beginners in programming, as early as CS1 and CS2(in Stanford vocabulary, CS106A and B).

•The choice of context can influence both students' motivation and quality of learning. Cooper et al. proposed to use the context of security in which to create CS1/2 laboratory exercises/modules. **My summer project is to create two of the secure coding lab modules on input validation and checking return values.**

7 Important Topics in Secure Coding

Laboratory exercises and associated serious games, each designed to teach a specific IA(information assurance) concept:

1. Validating user input
2. Array range checking
3. Buffer overflow

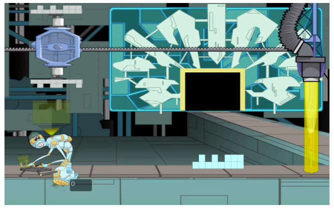

Fig. 2 Implementation of module 1. validating user input

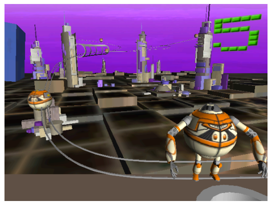

Fig. 3 Implementation of module 6. Returning values and Handling Errors

4. Operator precedence
5. Rounding errors
6. Returning values and handling errors
7. Numeric overflow/underflow

Return value checking in the context of Functions lab

•On failure, some functions return specific values as a signal.

•If users don't check those return values, the functions fail silently

•Examples include `pow(math.h)`, `strstr(string.h)`, `atoi (stdlib.h)`, and `snprintf(stdio.h)`.

•Such functions are integrated into exercises that force students to check for return values.

•An example from the functions lab uses the `pow` function to send encryption keys. Students are first given a faulty piece of code, later shown the fix:

```
// FAULTY CODE!
/* rsa_encryption.cpp */
#include <math.h>
#include <iostream>
using namespace std;

int main(){
     doublepublic_key,first_prime,second_prime;
     first_prime = pow(2,6972593)-1;
     second_prime = 17;
     public_key = first_prime*second_prime;
     cout << "Public key : " << public_key << endl;
     /*The rest goes here*/
     return 0;
}
// AFTER THE FIX (identical parts excluded)
……
     first_prime = pow(2,6972593)-1;
     second_prime = 17;
     while(first_prime == HUGE_VAL || second_prime ==
HUGE_VAL)
     {
        cout << "Enter two primes within range!" << endl;
        cin >> first_prime >> second_prime;
     }
        public_key = first_prime * second_prime;
……
```

Input validation in the context of String Lab

• Unvalidated input can cause havoc to a program. (ex. SQL injection)

• Input may come from the user or files, often in strings.

•Interesting examples of string related functions are sprintf from stdio.h, system from stdlib.h, and cin from iostream.

•`system` function invokes the command processor to execute a command. In this example, students are asked to fix the problem by using string functions to check for semicolons in the user input.

```
/*printer_injection.cpp*/
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
using namespace std;

int main(){
     string file_name;
     cout << "Please enter the file name: " << endl;
     cin >> file_name;
     char buf[1024];
     snprintf(buf,sizeof(buf)-1, "PRINT [/D:myprinter]
              %s", file_name.c_str());
     system(buf);
     return 0;
}
```

By concatenating a semicolon and a command, the user can take control of the system. When prompted to enter the file name:

```
myfile.txt;xterm // open up a new command window
myfile.txt;rm -rf //recursively remove everything on
the computer.
```

Next Steps

•Finish writing and revising the second lab
•Test the lab's effectiveness with students

This work is partially supported by NSF award DUE 1022557 - Building a serious game to teach secure coding in introductory programming. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.