Evaluating Secure Programming Knowledge

Matt Bishop, UC Davis
Jun Dai, Cal State Sacramento
Melissa Dark, Purdue University
Ida Ngambeki, Purdue University
Phillip Nico, Cal Poly San Luis Obispo
Minghua Zhu, UC Davis

Special thanks to: Somdutta Bose, UC Davis; Steven Belcher, NSA

“Secure” Programming

- Proper definition: programming designed to satisfy a security policy
  - But it is rarely used to mean this …
- Usual usage: programming designed to prevent problems that might cause security breaches
  - Hence “defensive programming” or “robust programming”

The Problem

- Software quality is poor … very poor
- Classroom teaching:
  - In regular classes, will crowd out existing content
  - Also, many faculty don't know (or don't care) about this; they focus on class content
  - In a class focusing on this, can't require all students to take it as schedules full
  - Also that won't help non-CS students!

Hey, how do I get to Carnegie Hall?

Practice, Madam, practice!

Look at Humanities, Social Sciences

- Think of a writing clinic!
  - English (and other) departments, law schools
  - Focus on grammar, writing style, organization
  - *And not content!*
- Analogue for computer science:
  - Focus on robust programming practice, coding style, organization
  - *And not correctness with respect to the assignment!*

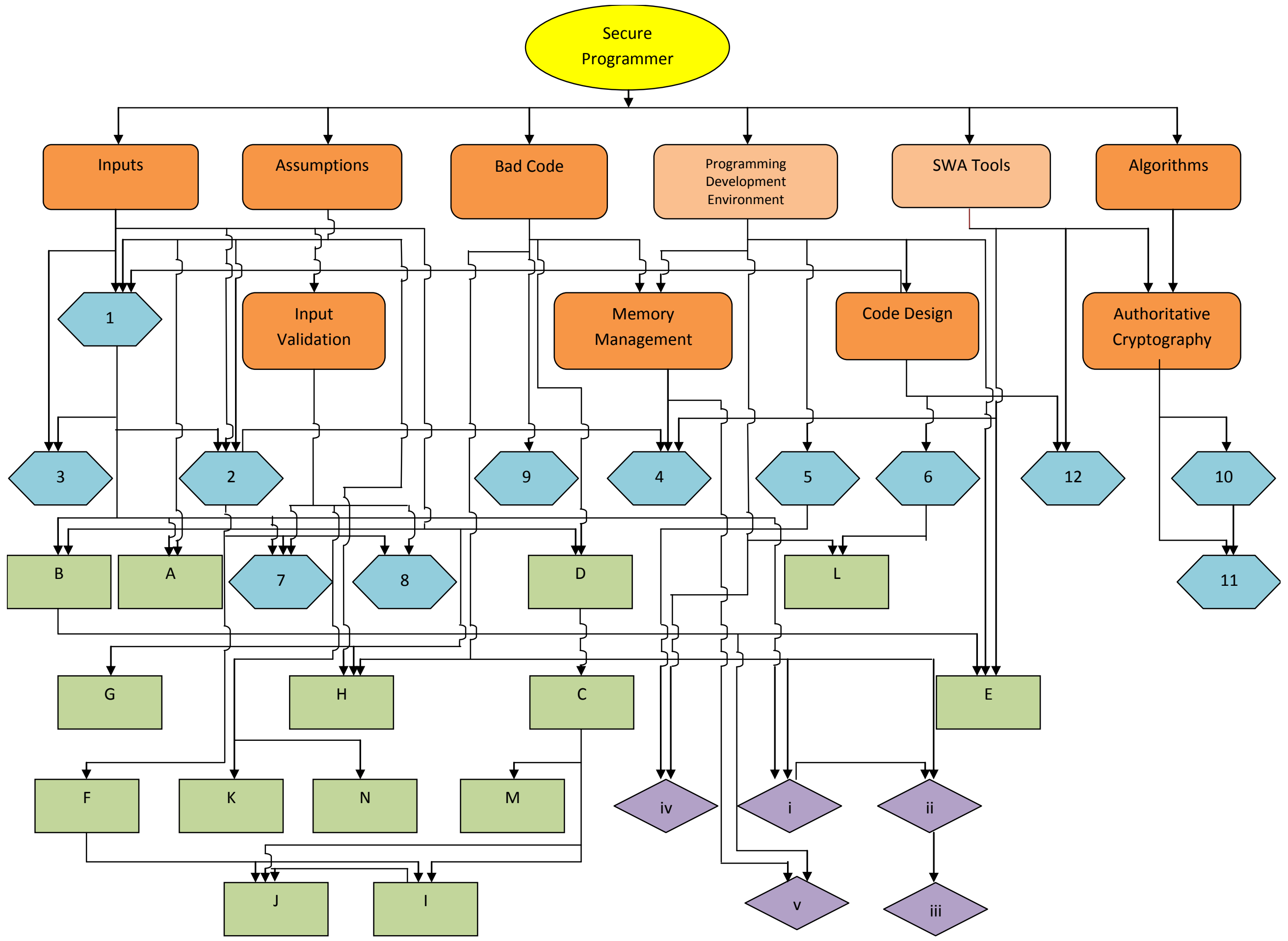
Benefits

- Students learn robust programming techniques through analysis of their own programs
  - Tools are good; students learn how to use them, how to interpret results
- Students learn robust programming techniques apply to all programs, not only to a specific class or assignment

How We Do This

- Understand how students think about robust programming
- Assess whether the clinic is having desired effect on student understanding of robust programming
  - Pre-clinic assessment test
  - Post-clinic assessment test

Concept Inventory



Concept Inventory Key

Very Important ⬣

1. Assume whatever can go wrong will
2. Assume any input is going to be malformed or not what you expect
3. Do not make a security decision based on un-trusted inputs
4. Check that all arguments are of the correct type and will not overflow any arrays
5. Use data abstraction to enable the compiler to perform rigorous type checking and to enforce constraints on values and lengths
6. Understand the context in which the program will execute
7. Validate your input stream to ensure that the commands invoked are expected and no other commands are injected
8. When performing input validation take into account how programs invoked with those arguments could interpret them
9. Avoid hard coded passwords and secrets in your program
10. Use well known and accepted cryptographic algorithms and. Don't use obsolete or deprecated cryptographic algorithms or create your own algorithms
11. Use well known and accepted cryptographic random number generation. Don't use obsolete or deprecated cryptographic algorithms or create your own algorithms
12. Many tools help you create a secure program, please take advantage of them

Somewhat Important ◆

i. Hide details that users don't need to know about
ii. Avoid side effects in arguments to unsafe macros. If a developer is using a macro that uses its arguments more than once, then the developer must avoid passing any arguments with side effects to that macro
iii. Use parentheses around macro replacement lists. Otherwise operator precedence may cause the expression to be computed in unexpected ways
iv. Minimize the scope of variables and functions. This prevents many unexpected changes to the variables due to programming error
v. When the memory a pointer points to is freed, set the pointer to NULL. Otherwise, these dangling pointers could cause writing to freed memory, and create a double free vulnerability.

Important 🟩

A. If you have no reason to trust it, don't trust it. Take greater care with any input you have not generated
B. If it cannot happen, check for it. Someone may modify the program in such a way that it can happen ... or you may be wrong
C. Do not use input or constructor string functions that do not perform any bound checking
D. Do not use input or constructor functions that cannot check the length of the input
E. C and C++ compilers generally do not check types rigorously. A developer can increase this level of checking by turning on compiler warnings, which will often catch more type errors than if they are not used
F. Avoid calls to malloc() with the parameter (number of bytes to be allocated) set to 0. Either the function returns NULL, or it returns a pointer to space that cannot be used without overwriting unallocated memory
G. Control the input values when possible by limiting them to a finite set
H. Calling functions with null parameters for input should be checked for and defended against
I. Type conversion issues especially for cases that may result in integer wraparound and overflows
J. Rules for pointer arithmetic as vulnerabilities can arise when addition or size checks involve two pointer types
K. When performing input validation make sure that any validated path does not allow escaping from a restricted directory
L. Before creating a directory or file, make sure you have set the correct default permission specification
M. Be wary of off by one errors
N. When using format string functions, make sure that the format string can be authenticated/trusted

Example Question: Handling User Input

Concept: “If you have no reason to trust it, don't trust it. Take greater care with any input you have not generated.”

Question: User input can be unpredictable. Which of the following is the best way to avoid problems processing that input?

a) Elevate privileges when processing user-provided input, to ensure the computation can be done.
b) Drop unnecessary privileges when processing user-provided input, to limit the effects of bad user input.
c) Keep privileges constant whenever possible, for more readable code that is easier to maintain without introducing error.
d) Assign elevated privileges to a new process or thread that reads the input and does the computation, so that any malicious side-effects do not affect the primary process or thread.
e) Keep privileges the same but constrain the process execution in a sandbox so that any malicious side-effects are contained.

Example Question: Handling User Input

Question: User input can be unpredictable. Which of the following is the best way to avoid problems processing that input?

a) Elevate privileges when processing user-provided input, to ensure the computation can be done. 5% chose this

b) Drop unnecessary privileges when processing user-provided input, to limit the effects of bad user input. 18% chose this

c) Keep privileges constant whenever possible, for more readable code that is easier to maintain without introducing error. 9% chose this

d) Assign elevated privileges to a new process or thread that reads the input and does the computation, so that any malicious side-effects do not affect the primary process or thread. 43% chose this

e) Keep privileges the same but constrain the process execution in a sandbox so that any malicious side-effects are contained. 25% chose this

Example Question: Indexing Into an Array

Concept: “Check parameters to ensure that all arguments are of the correct type and will not overflow any arrays.”

Question: Your program accepts parameters `x`, `y`, and `z` to calculate the position of an item in an array relative to the current item indexed by `ptr`.

```
101 newOffset = (x*colSize)+(y*rowSize)–z;
102 ptr = ptr + newOffset;
103 newObject = objectArray[ptr];
```

Which of the following is true?

a) I should check that the result in line 101 is not negative.

b) I should check that the result in line 101 is not null.

c) I should check that the result in line 102 is not negative.

d) I should check that the result in line 102 is not null.

Example Question: Indexing Into an Array

Question: Your program accepts parameters `x`, `y`, and `z` to calculate the position of an item in an array relative to the current item indexed by `ptr`.

```
101 newOffset = (x*colSize)+(y*rowSize)–z;
102 ptr = ptr + newOffset;
103 newObject = objectArray[ptr];
```

Which of the following is true?

a) I should check that the result in line 101 is not negative. 28% chose this

b) I should check that the result in line 101 is not null. 17% chose this

[✓ c) I should check that the result in line 102 is not negative. 38% chose this]

d) I should check that the result in line 102 is not null. 17% chose this

Example Question: Handling Missing Data

Concept: “If you have no reason to trust it, don't trust it. Take greater care with any input you have not generated.”

Question: You must read a list of user names and starting date: day, month, year. Then your program must sort them in ascending order to create a list of users by seniority. Some start dates are missing the day or month of the start date. This list-sorting function may be used elsewhere, or tweaked in the future. Which statement below is the most robust way to handle the missing data?

a) Initialize the variables for missing information with a random plausible value.
b) Leave the variables for missing information uninitialized.
c) Initialize the variables for missing information with 0.
d) Initialize the variables for missing information with the maximum plausible value.

Example Question: Handling Missing Data

Concept: “If you have no reason to trust it, don't trust it. Take greater care with any input you have not generated.”

Question: You must read a list of user names and starting date: day, month, year. Then your program must sort them in ascending order to create a list of users by seniority. Some start dates are missing the day or month of the start date. This list-sorting function may be used elsewhere, or tweaked in the future. Which statement below is the most robust way to handle the missing data?

a) Initialize the variables for missing information with a random plausible value. 7% chose this

b) Leave the variables for missing information uninitialized. 13% chose this

c) Initialize the variables for missing information with 0. 57% chose this

d) Initialize the variables for missing information with the maximum plausible value. 22% chose this

Example Question: Pointer Validation

Concepts: “Follow the rules for pointer arithmetic as vulnerabilities can arise when addition or size checks involve two pointer types” and “Be wary of off by one errors.”

Question: For a C program you must create an array of `size` integers. You write:

```
unsigned long *start, *end;
start = malloc(size*sizeof (unsigned long));
```

Assuming `malloc` succeeds, the correct value for end can be computed by:

```
a) end = start + size * sizeof(unsigned long);
b) end = start + size * (sizeof(unsigned long) - 1);
c) end = start + (size - 1) * sizeof(unsigned long);
d) end = start + size – 1;
e) end = start + sizeof(unsigned long) – 1;
```

Example Question: Pointer Validation

Concepts: "Follow the rules for pointer arithmetic as vulnerabilities can arise when addition or size checks involve two pointer types" and "Be wary of off by one errors."

Question: For a C program you must create an array of `size` integers. You write:

```
unsigned long *start, *end;
start = malloc(size*sizeof (unsigned long));
```

Assuming `malloc` succeeds, the correct value for `end` can be computed by:

a) `end = start + size * sizeof(unsigned long);` 10% chose this

b) `end = start + size * (sizeof(unsigned long) - 1);` 16% chose this

c) `end = start + (size - 1) * sizeof(unsigned long);` 31% chose this

d) `end = start + size – 1;` 20% chose this

e) `end = start + sizeof(unsigned long) – 1;` 16% chose this

Example Question: Input Validation

Concepts: “Assume any input is going to be malformed or not what you expect.”

Question: You must write a function that stores an integer in the destination pointed to by value, and returns an integer indicating success or failure. You start with this function prototype:

```
int getSeconds (int * secondsParameter )
```

Which of the following must you do before or instead of any of the others?

a) I must dereference the pointer to get the memory location.

b) I must find the value that the pointer refers to.

c) I must check that the pointer passed in does not already have a value.

d) I must check that the pointer passed in is not **NULL**.

Example Question: Input Validation

Concepts: "Assume any input is going to be malformed or not what you expect."

Question: You must write a function that stores an integer in the destination pointed to by value, and returns an integer indicating success or failure. You start with this function prototype:

```
int getSeconds (int * secondsParameter )
```

Which of the following must you do before or instead of any of the others?

a) I must dereference the pointer to get the memory location. 10% chose this

b) I must find the value that the pointer refers to. 8% chose this

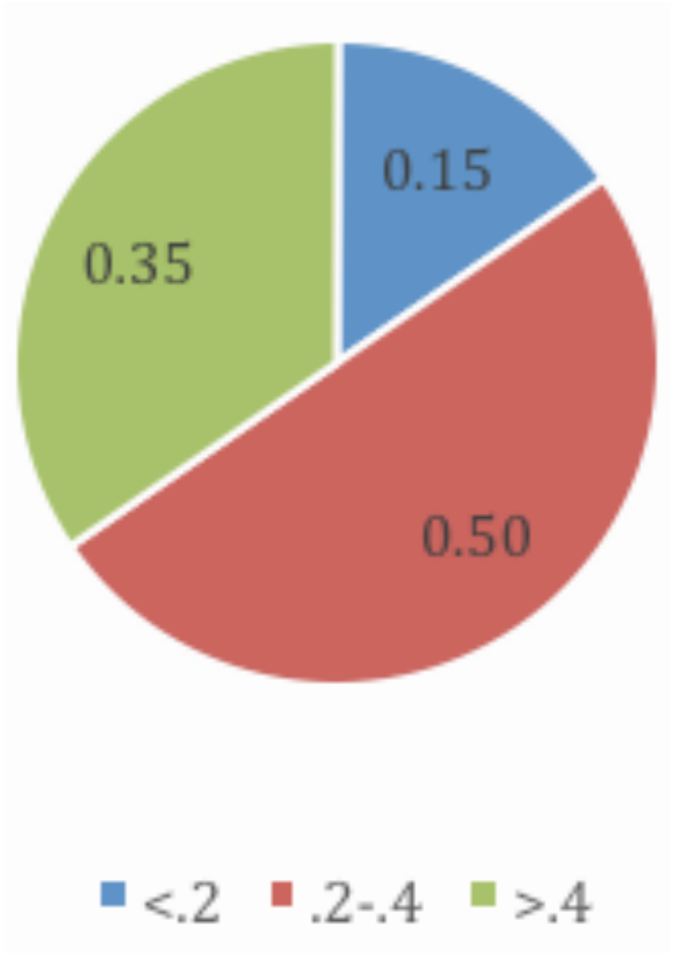
c) I must check that the pointer passed in does not already have a value. 13% chose this

d) I must check that the pointer passed in is not **NULL**. 69% chose this

Analysis

- Analyze test question, distractors
  - Item effect: which students with a high overall score got a particular question correct
    - –1.00 to 1.00
  - Identifies questions that are not functioning, ie. low or negative correlation with overall score
  - This implies distractors confuse students who know the material

Analysis



Conclusion

- Evaluation of distractors important to be able to measure effectiveness of secure programming clinic
- So far, 3 schools involved in the clinic, and it has been run for 3 different classes
- Thus far, clinic seems to be effective
  - But we're still gathering data …

Weinberg's Second Law

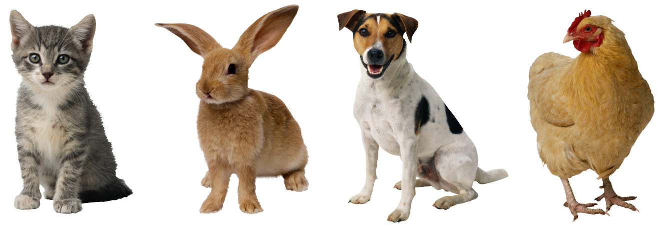
If builders built buildings the way programmers wrote programs ...

the first woodpecker to come along would destroy civilization

Thanks To

Funded by NSF under Grant No. DGE-1303048 to Purdue University and Grant No. DGE-1303211 to the University of California at Davis

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the universities involved, or any of our pets



Presenter

Matt Bishop
Department of Computer Science
University of California at Davis
1 Shields Ave.
Davis, CA 95616-8562
USA

email: mabishop@ucdavis.edu
phone +1 (530) 752-8060
web: http://seclab.cs.ucdavis.edu/~bishop