Teaching the Security Mindset to CS 1 Students

Vahab Pournaghshband
University of California, Los Angeles
vahab@cs.ucla.edu
Overview

▶ Motivation

▶ What is Security Mindset?

▶ Advantages

▶ Methodology

▶ Evaluation

▶ Summary
Java Flaw Puts Millions Of Windows And Mac Users At Risk

A new and serious vulnerability found in the Java platform that is installed into millions of browsers is under attack from hackers.

"Note: If you're not interested in reading about security vulnerabilities and just want to know what to do to be safe, jump to the last paragraph.

Citi: Millions stolen in May hack attack

Cyberattacks on Iran — Stuxnet and Flame

Updated: Aug. 9, 2012

Over the last few years, Iran has become the target of a series of notable cyberattacks, some of which were linked to its nuclear program. The best known of these was Stuxnet, the name given to a computer worm, or malicious computer program.

According to an article in The New York Times in June 2012, during President Obama's first few months in office, he was reportedly...
“Security requires a particular mindset. Security professionals see the world differently... This kind of thinking is not natural for most people. It’s not natural for engineers. Good engineering involves thinking about how things can be made to work; the security mindset involves thinking about how things can be made to fail. It involves thinking like an attacker, an adversary or a criminal. You don’t have to exploit the vulnerabilities you find, but if you don’t see the world that way, you’ll never notice most security problems.”

Bruce Schneier
Advantages

- Early exposure to security issues
- Students are more prone to naturally have security in mind
- Helps preventing undesirable habits of overlooking security
- Learning the significance of security bugs
- Minor logical or run-time errors can lead to security breaches
- "Security can make other stuff more interesting"
- Incorporating simple attack/defense scenarios to teach and verify program correctness interactively
Advantages

▶ Early exposure to security issues
  ▶ Students are more prone to naturally have security in mind

▶ Minor logical or run-time errors can lead to security breaches
▶ “Security can make other stuff more interesting”
▶ Incorporating simple attack/defense scenarios to teach and verify program correctness interactively
Advantages

▶ Early exposure to security issues
  ▶ Students are more prone to naturally have security in mind
  ▶ Helps preventing undesirable habits of overlooking security bugs
  ▶ Learning the significance of security bugs
  ▶ Minor logical or run-time errors can lead to security breaches
  ▶ “Security can make other stuff more interesting”
  ▶ Incorporating simple attack/defense scenarios to teach and verify program correctness interactively
Advantages

▶ Early exposure to security issues
  ▶ Students are more prone to naturally have security in mind
  ▶ Helps preventing undesirable habits of overlooking security bugs

▶ Learning the significance of security bugs
Advantages

- Early exposure to security issues
  - Students are more prone to naturally have security in mind
  - Helps preventing undesirable habits of overlooking security bugs

- Learning the significance of security bugs
  - Minor logical or run-time errors can lead to security breaches
Advantages

▶ Early exposure to security issues
  ▶ Students are more prone to naturally have security in mind
  ▶ Helps preventing undesirable habits of overlooking security bugs
▶ Learning the significance of security bugs
  ▶ Minor logical or run-time errors can lead to security breaches
▶ “Security can make other stuff more interesting”
Advantages

- Early exposure to security issues
  - Students are more prone to naturally have security in mind
  - Helps preventing undesirable habits of overlooking security bugs

- Learning the significance of security bugs
  - Minor logical or run-time errors can lead to security breaches

- “Security can make other stuff more interesting”
  - Incorporating simple attack/defense scenarios to teach and verify program correctness interactively
Advantages

- Non-CS majors take only CS 1 & 2
Advantages

- Non-CS majors take only CS 1 & 2
  - They understand the root cause of known security problems
Advantages

- Non-CS majors take only CS 1 & 2
  - They understand the root cause of known security problems
  - “The security mindset is a valuable skill that everyone can benefit from, regardless of career path”
Advantages

▶ Non-CS majors take only CS 1 & 2
  ▶ They understand the root cause of known security problems
  ▶ “The security mindset is a valuable skill that everyone can benefit from, regardless of career path”
▶ Secure programming takes extensive practice to evolve into a skill
Advantages

▶ Non-CS majors take only CS 1 & 2
  ▶ They understand the root cause of known security problems
  ▶ “The security mindset is a valuable skill that everyone can benefit from, regardless of career path”

▶ Secure programming takes extensive practice to evolve into a skill
  ▶ Failure of a single course in computer security in undergraduate curriculum
Methodology

- The Login Program

- Asks for username and password and reveals a secret if they are correct
- Generates an error message if either one is incorrect

Why this example?
- Basic concept is familiar to students
- Can be very simple to very complex
- Can be incrementally built up throughout the term
- It is security-sensitive by its nature
Methodology

- The Login Program
  - Asks for username and password and reveals a secret if they are correct
Methodology

The Login Program

- Asks for username and password and reveals a secret if they are correct
- Generates an error message if either one is incorrect
Methodology

▶ The Login Program
  ▶ Asks for username and password and reveals a secret if they are correct
  ▶ Generates an error message if either one is incorrect

▶ Why this example?
Methodology

- The Login Program
  - Asks for username and password and reveals a secret if they are correct
  - Generates an error message if either one is incorrect

- Why this example?
  - Basic concept is familiar to students
Methodology

▶ The Login Program

▶ Asks for username and password and reveals a secret if they are correct

▶ Generates an error message if either one is incorrect

▶ Why this example?

▶ Basic concept is familiar to students

▶ Can be very simple to very complex
Methodology

▶ The Login Program

▶ Asks for username and password and reveals a secret if they are correct

▶ Generates an error message if either one is incorrect

▶ Why this example?

▶ Basic concept is familiar to students

▶ Can be very simple to very complex

▶ Can be incrementally built up throughout the term
Methodology

- The Login Program
  - Asks for username and password and reveals a secret if they are correct
  - Generates an error message if either one is incorrect

- Why this example?
  - Basic concept is familiar to students
  - Can be very simple to very complex
  - Can be incrementally built up throughout the term
  - It is security-sensitive by its nature
Conditional (if-else) Statements

```cpp
if ( password == 12345 )
    cout << "The secret word is Peace.";
else
    cout << "Invalid password!";
```
Conditional (if-else) Statements

```cpp
if ( password == 12345 )
    cout << "The secret word is Peace.";
else
    cout << "Invalid password!";
```

String Class
_nested if-else and switch Statements

```cpp
if ((username == "vahab" || password == "5!eR?3") ||
    (username == "peter" || password == "0a%2NFa"))
    cout << "The secret word is Peace."
else
    cout << "Invalid username and/or password!"
```
Methodology: Examples

- Loops
  - Multiple login attempts to protect from brute force attack
  - Logical errors in loop condition can be catastrophic (e.g., infinite loop)

- File I/O
  - Hardcoding passwords into executables is bad security practice
  - Introducing credentials file: to add, remove, and update users' passwords

- Touch upon cryptographically secure one-way hash functions (e.g., crypt)
Methodology: Examples

- **Loops**
  - Multiple login attempts to protect from brute force attack
Methodology: Examples

- **Loops**
  - Multiple login attempts to protect from brute force attack
  - Logical errors in loop condition can be catastrophic (e.g., infinite loop)
Methodology: Examples

- **Loops**
  - Multiple login attempts to protect from brute force attack
  - Logical errors in loop condition can be catastrophic (e.g., infinite loop)

- **File I/O**
  - Hardcoding passwords into executables is bad security practice
  - Introducing credentials file: to add, remove, and update users' passwords
  - Touch upon cryptographically secure one-way hash functions (e.g., crypt)
Methodology: Examples

- **Loops**
  - Multiple login attempts to protect from brute force attack
  - Logical errors in loop condition can be catastrophic (e.g., infinite loop)

- **File I/O**
  - Hardcoding passwords into executables is bad security practice
Methodology: Examples

- **Loops**
  - Multiple login attempts to protect from brute force attack
  - Logical errors in loop condition can be catastrophic (e.g., infinite loop)

- **File I/O**
  - Hardcoding passwords into executables is bad security practice
  - Introducing *credentials file*: to add, remove, and update users’ passwords
Methodology: Examples

- **Loops**
  - Multiple login attempts to protect from brute force attack
  - Logical errors in loop condition can be catastrophic (e.g., infinite loop)

- **File I/O**
  - Hardcoding passwords into executables is bad security practice
  - Introducing *credentials file*: to add, remove, and update users’ passwords
  - Touch upon cryptographically secure one-way hash functions (e.g., crypt)
Arrays and C Strings

```cpp
cchar password[SIZE];
bool logged_in = false;

cin >> password;

if ( strcmp(password,correct_password) == 0 )
    logged_in = true;

if ( logged_in == true )
    cout << "The secret word is Peace.";
else
    cout << "Invalid password!";
```
class Credentials {
public:
    Credentials(char* filename);
    bool addNewUser(char*, char*, char*, char*, int, long, char*, char*, char);
    int deleteUser(UserInfo*);
    UserInfo* getUserInfo(char* username);
private:
    int num_of_users;
    UserInfo* users_list;
};

class UserInfo {
public:
    UserInfo(char*, char*, char*, char*, int, long, char*, char*, char);
    char* getUsername();
    char* getPassword();
    bool resetPassword(char* password);
    bool isStrongPassword(char* password);
    char first_name[SIZE];
    char last_name[SIZE];
    int age;
    long dob;
    char security_question[SIZE];
    char answer_security_question[SIZE];
    char privileges;
private:
    char username[SIZE];
    char password[SIZE];
};
<table>
<thead>
<tr>
<th>Topics</th>
<th>Login Program</th>
<th>Other Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H</td>
<td>D</td>
</tr>
<tr>
<td>Intro/Variables</td>
<td>3.2</td>
<td>1.4</td>
</tr>
<tr>
<td>if-else Statements</td>
<td>3.8</td>
<td>2.0</td>
</tr>
<tr>
<td>Nested if-else</td>
<td>4.3</td>
<td>2.4</td>
</tr>
<tr>
<td>String Class</td>
<td>3.3</td>
<td>2.1</td>
</tr>
<tr>
<td>Loops</td>
<td>4.1</td>
<td>2.7</td>
</tr>
<tr>
<td>Nested Loops</td>
<td>3.4</td>
<td>3.5</td>
</tr>
<tr>
<td>Functions</td>
<td>3.2</td>
<td>3.1</td>
</tr>
<tr>
<td>Arrays/C strings</td>
<td>3.7</td>
<td>3.2</td>
</tr>
<tr>
<td>Multi-dim Arrays</td>
<td>3.2</td>
<td>3.6</td>
</tr>
<tr>
<td>Parallel Arrays</td>
<td>3.8</td>
<td>3.0</td>
</tr>
<tr>
<td>File I/O</td>
<td>3.6</td>
<td>4.0</td>
</tr>
<tr>
<td>Pointers/DMA</td>
<td>3.6</td>
<td>4.0</td>
</tr>
<tr>
<td>Structs/Classes</td>
<td>4.2</td>
<td>4.3</td>
</tr>
</tbody>
</table>
Table: Student evaluation of materials we used in CS 1. (H: Helpfulness, D: Difficulty, F: Fun)

<table>
<thead>
<tr>
<th>Topics</th>
<th>Login Program</th>
<th>Other Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro/Variables</td>
<td>3.2 1.4 3.1</td>
<td>3.0 1.4 2.1</td>
</tr>
<tr>
<td>if-else Statements</td>
<td>3.8 2.0 3.9</td>
<td>4.0 2.1 3.1</td>
</tr>
<tr>
<td>Nested if-else</td>
<td>4.3 2.4 4.6</td>
<td>3.3 2.6 3.0</td>
</tr>
<tr>
<td>String Class</td>
<td>3.3 2.1 3.8</td>
<td>3.5 1.7 3.3</td>
</tr>
<tr>
<td>Loops</td>
<td>4.1 2.7 4.2</td>
<td>3.8 2.7 3.4</td>
</tr>
<tr>
<td>Nested Loops</td>
<td>3.4 3.5 3.7</td>
<td>3.9 3.2 3.8</td>
</tr>
<tr>
<td>Functions</td>
<td>3.2 3.1 3.9</td>
<td>3.9 3.4 3.7</td>
</tr>
<tr>
<td>Arrays/C strings</td>
<td>3.7 3.2 4.0</td>
<td>3.7 2.9 3.8</td>
</tr>
<tr>
<td>Multi-dim Arrays</td>
<td>3.2 3.6 4.1</td>
<td>3.5 3.4 2.9</td>
</tr>
<tr>
<td>Parallel Arrays</td>
<td>3.8 3.0 4.4</td>
<td>3.7 3.1 3.2</td>
</tr>
<tr>
<td>File I/O</td>
<td>3.6 4.0 4.5</td>
<td>3.4 3.4 3.1</td>
</tr>
<tr>
<td>Pointers/DMA</td>
<td>3.6 4.0 4.5</td>
<td>3.5 3.9 2.8</td>
</tr>
<tr>
<td>Structs/Classes</td>
<td>4.2 4.3 4.2</td>
<td>3.7 4.2 3.5</td>
</tr>
</tbody>
</table>

Vahab Pournaghshband | Teaching the Security Mindset to CS 1 Students
Summary

- Lack of security mindset is responsible for many overlooked bugs
- Teaching the security mindset is valuable and effective
- Teaching by example: the login program
- Positive reaction by students
- This is just the first step...