Secure Programming in C/C++

Kenneth Ingham

September 29, 2009

1 Course overview

This class is for C and C++ programmers who want to write code with fewer exploitable security bugs. The class focuses on the practice of C coding, and should be applicable to all software development models (e.g., agile development, the waterfall model, etc).

2 Course objectives

- Learn how to recognize and avoid common C and C++ coding errors, that can lead to exploitable bugs, including:
  - problems with integers, including value truncation, signed and unsigned mixing, overflow, and underflow.
  - format string errors.
  - buffer overflows on the stack and heap.
  - memory management issues (other than buffer overflow).
  - race conditions.
- Learn the security issues that apply only to C++.
- Learn about re-entrant code and why it is important for signal handling.
- Learn the importance and difficulty of erasing sensitive data.

3 Student background

If you are attending this class, then we assume that

- You have attended the "Building Secure Systems" class to understand that half of security problems are design flaws, and that you cannot code your way out of a bad design.

- You have working knowledge of C and/or C++.

Initial versions of the class will be run on Linux, which means that the students need to be familiar with an editor and file manipulation on Linux.

4 Logistics

The class lasts two days. The student computers need run Linux; eventually the class will work with Windows as well. The class uses the following software:

- development tools such as *make*
- Gnu C/C++ compiler
- Gnu C/C++ compiler (on Linux distribution but needed for Windows)
- Internet access
- Missing from C-coding.tex
- PhkMalloc (in class files)
- *gdb* (on Linux distribution but needed for Windows)
- *nm* (on Linux distribution)
- *objdump* (on Linux distribution but needed for Windows)
- *pscan*
- *rats* (on class web site)
- development tools such as *make*
- development tools such as *make* (on Linux distribution but needed for Windows)
- development tools such as *make* (on Linux distribution)
- electric fence (in class files)
- the GD library and include files (gd-devel)
- the PNG library and its include files
- valgrind

This class has no special network requirements beyond general Internet access.

The class needs a web server for the class web site. The instructor's laptop may be this web server; otherwise the machine provided in the classroom for the instructor is a good choice. This machine obviously will need web server software installed.

5 Class outline

1. Introduction (Lecture: 15; Lab: 0)
   (a) Class Introductions
   (b) Class Logistics
      i. Class schedule
      ii. Breaks
      iii. Question policy

        iv. Break room and restroom locations
        v. Assumptions about your background
    (c) Typographic conventions
2. Secure C/C++ Programming (Lecture: 15; Lab: 0)
    (a) A real exploit program
    (b) Security is a process, not a product
    (c) Adding cryptography does not make a program secure
    (d) C++ versus C
    (e) Summary
3. Security and the software development life cycle (Lecture: 25; Lab: 20)
    (a) Introduction
    (b) Requirements
        i. Example
        ii. Use, Abuse, and Misuse cases
    (c) Design/Architecture
        i. Design is critical
        ii. Properly-written specifications
    (d) Code development
        i. Implementation is critical
    (e) Testing
    (f) Operations/maintenance
    (g) Agile development
    (h) Penetrate and patch is the wrong approach
    (i) Summary
    (j) Lab
4. Integer operations (Lecture: 40; Lab: 75)
    (a) Introduction
    (b) Value truncation
        i. Example
        ii. Example
        iii. How can you avoid this problem?
    (c) Integer overflow and underflow
        i. Examples
        ii. How can you avoid this problem?
    (d) Signed/unsigned problems
        i. Example
        ii. How can you avoid this problem?
    (e) Non-exceptional conditions
    (f) Summary

(g) Lab

5. Buffer overflow introduction (Lecture: 20; Lab: 0)
   (a) Introduction
   (b) A simple buffer overflow example
   (c) Memory layout
      i. Example
   (d) Summary
6. Stack overflows (Lecture: 20; Lab: 60)
   (a) Introduction
   (b) Exploit: calling another function
   (c) Other exploits for stack overflows
   (d) More stack overflow information
   (e) A real stack overflow exploit program
   (f) Avoiding these problems
   (g) Summary
   (h) Lab
7. Heap and other data segment overflows (Lecture: 15; Lab: 40)
   (a) Introduction
      i. Example
   (b) The heap
      i. Example
   (c) Heap data structure attacks
   (d) Common programming errors
   (e) Example
   (f) Avoiding these problems
   (g) Summary
   (h) Lab
8. Pointer issues (Lecture: 35; Lab: 50)
   (a) Introduction
      i. Example
   (b) Pointers to functions
   (c) C++ virtual method table
      i. Example
      ii. Destructors
   (d) Global offset table (GOT)
   (e) The `.dtors` section
      i. Example
   (f) Exit handlers

        i. Example
    (g) **setjmp/longjmp**
    (h) Exception handling
    (i) Avoiding these problems
    (j) Summary
    (k) Lab
9. Buffer overflow avoidance and mitigation (Lecture: 45; Lab: 45)
    (a) Standard C library functions
        i. Example
        ii. Alternate functions/libraries
    (b) **malloc** aids
        i. PhkMalloc
        ii. Electric fence
    (c) Die Hard
    (d) *valgrind*
        i. Example
    (e) Programming strategies
    (f) OS- and compiler-level countermeasures
        i. General
        ii. Stack
        iii. Heap
    (g) Summary
    (h) Lab
10. Format string errors (Lecture: 25; Lab: 60)
    (a) Introduction
        i. Example
    (b) It gets worse!
    (c) A real attack program
    (d) The problem
    (e) How can you avoid this problem?
    (f) Summary
    (g) Lab
11. Tips and techniques (Lecture: 35; Lab: 45)
    (a) Fail securely
        i. Example
    (b) Validate *all* input
    (c) Only output safe data
    (d) C++ class libraries and templates
    (e) Exception handling

(f) Complete coverage
    i. Example
(g) Tools
    i. Static analysis tools
    ii. Dynamic analysis tools
    iii. Testing tools
(h) Summary
(i) Lab

12. Process issues (Lecture: 20; Lab: 25)
    (a) Erasing data
        i. The problem
        ii. Some solutions
    (b) Signals
        i. Signal-safe calls
    (c) Signal race conditions
        i. How to avoid these problems
    (d) Summary
    (e) Lab
13. Filesystem Security Issues (Lecture: 20; Lab: 25)
    (a) Erasing data in the filesystem
        i. The problem
        ii. Solutions
            A. Encrypting on-disk data
    (b) Filesystem TOCTOU race conditions
        i. Example: *passwd* command races
        ii. Avoiding TOCTTOU problems
        iii. Coding tips to reduce problems
        iv. Example of **open** with `O_EXCL` and `O_CREAT`
    (c) Temporary Files
        i. Coding tips to reduce problems
    (d) Summary
    (e) Lab