Designing and Coding Secure Systems

Kenneth Ingham and Anil Somayaji

September 29, 2009

1 Course overview

This class covers secure coding and some design issues from a language neutral approach—you can make mistakes such as input validation or failing to use defense in depth in any language. The course stresses how to avoid security problems through the proper implementation of programs. This class makes heavy use of labs where the instructor presents a case study and the students discuss how to apply the concepts presented to the example under discussion; the example can also be a system in which the students are involved.

This class is appropriate for students who are programmers; you cannot code your way out of a bad design, and recognizing design flaws earlier allows them to be fixed with fewer resources. This class is also appropriate for program designers and system architects; they need to understand how to design in security from the beginning.

2 Course objectives

Students attending this class will learn:

- Why we have had problems writing secure code.
- Security belongs in all phases of the software development lifecycle.
- What is a threat model and how does it help with security.
- The design principle of secure failure modes.
- Important language-independent coding issues such as input validation.
- Concepts of authorization and access control with references to the Java and .NET libraries.
- How, what, when, why, and where to log.
- Issues affecting web server security.

- How to perform a software security audit.
- How to test for security.

3 Student background

If you are attending this class, then we assume that

- You are familiar with programming in a major language (e.g., Java, C, C++, C#, etc).
- You are a programmer, software designer, or system architect.
- You want to produce more secure systems.

4 Logistics

The class lasts two days. The student computers can run either Windows or Linux. The class uses the following software:

- Firefox Web Developer toolbar
- IE 6 or 7 (on Windows distribution) or Firefox 2 (on Linux distribution but needed for Windows) (optional)
- Internet access (optional)
- Java JDK 1.6 (for WebGoat)
- PortSwigger burp suite
- WebGoat v5.2
- a web browser

Since this is a discussion class, computers are not absolutely necessary. If computers are available, the student computers should have Internet access, including the necessary configuration for this to work. However, an isolated network can also work. If the classroom has computers, a web server is needed.

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. Designing and coding secure programs (Lecture: 50; Lab: 30)
   - (a) What is a secure program?
   - (b) Common security myths
     - i. “We have a firewall”
     - ii. “I use anti-virus software”
       - A. Example
   - (c) Why is Security Important?
   - (d) The Challenge
   - (e) Secure code is more reliable code
   - (f) Security versus usability
   - (g) Summary
   - (h) Lab
3. Threat models and risk management (Lecture: 45; Lab: 55)
   - (a) Introduction
     - i. Example
   - (b) The threat model
   - (c) The assets you are protecting
   - (d) Attackers
   - (e) Common attack goals
   - (f) Mitigating threats
   - (g) Examples
     - i. Password vault
     - ii. Web-based timesheet
   - (h) Risk analysis
   - (i) Failures of Imagination
   - (j) Summary
   - (k) Lab
4. Security and the software development life cycle (Lecture: 35; Lab: 20)
   - (a) Introduction
   - (b) Requirements
     - i. Example
     - ii. Example
     - iii. 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

5. Input validation and representation (Lecture: 60; Lab: 35)
    (a) Introduction
        i. Example input validation problems
    (b) Never trust the client
        i. Example: 3D3.Com ShopFactory
        ii. Example: Smartwin Technology CyberOffice Shopping Cart
    (c) Never trust the server
    (d) Never trust other programmers
        i. Example: Linux kernel
    (e) Beware Hidden User Input
    (f) Solution: Whitelists
    (g) Solution: Canonicalization
        i. Example: NTFS
        ii. Example: IIS and Nimda
    (h) Solution: Taint tracking
    (i) Summary
    (j) Lab

6. Fail securely (Lecture: 25; Lab: 15)
    (a) Introduction
    (b) Failure-related code is often poorly written and tested
        i. Examples
    (c) Proper failure state
    (d) Complete error/exception handling
        i. Example: Linux ELF binary loader
    (e) Resource issues
    (f) Failures take unexpected paths in the code
    (g) Backwards Compatibility
        i. Example: Windows file sharing
    (h) Reporting Errors Securely

   (i) Failing Functionally yet Securely
   (j) Summary
   (k) Lab

7. Logging (Lecture: 30; Lab: 35)
   (a) The purposes of logs
   (b) What to log
   (c) How to log
   (d) Resource issues
   (e) Distributed logging
   (f) Log monitoring
   (g) Sensitive data
   (h) Summary
   (i) Lab

8. State and the web (Lecture: 25; Lab: 65)
   (a) Overview
   (b) Ways of tracking state
      i. Hidden fields in forms
      ii. Cookies
      iii. CGI parameters
      iv. HTTP Referer field
   (c) Session hijacking
   (d) Solutions
   (e) Example vulnerable code
   (f) Testing for this problem
   (g) Summary
   (h) Lab

9. Code reviews (auditing) for security (Lecture: 45; Lab: 30)
   (a) Why to audit
   (b) How to audit
   (c) Automated code analysis tools
   (d) Items for an audit checklist
      i. Overall requirements and architecture issues
      ii. Defense in Depth
      iii. Compartmentalization
      iv. Privileges
      v. Authentication
      vi. Authorization
      vii. Cryptography
      viii. Secure failure
      ix. Input validation

- x. Logging
- xi. Sensitive data
- xii. Race conditions
- xiii. Scripting and extensibility
- xiv. State and the web
- xv. Deployment

(e) Summary

(f) Lab

10. Software testing for security (Lecture: 20; Lab: 30)
    - (a) Why to test
    - (b) Who should perform the testing
    - (c) How to test
    - (d) When to test
    - (e) Penetration testing and tools
    - (f) Limitations of Testing
    - (g) Summary
    - (h) Lab

Appendices

- A. Defense in depth (Lecture: 20; Lab: 25)
  - (a) Introduction
  - (b) Example: Web-based timesheet application
  - (c) Summary
  - (d) Lab
- B. Least privilege (Lecture: 40; Lab: 20)
  - (a) Introduction
  - (b) Dropping privileges after obtaining a resource
  - (c) Separation of privilege
    - i. Implementing separation of privilege
    - ii. Example: OpenSSH
      - A. Applying separation of privilege to OpenSSH
    - iii. Example: NTP client
  - (d) Example: Applying least privilege to the web timesheet application
  - (e) Summary
  - (f) Lab
- C. Compartmentalization (Lecture: 35; Lab: 20)
  - (a) Introduction
  - (b) Processes
  - (c) Filesystem Access Controls

    (d) Mandatory Access Control
    (e) Virtualization
        i. Hardware Virtualization
            A. Paravirtualization
        ii. Operating System-Level Virtualization
        iii. Application Virtualization
    (f) Summary
    (g) Lab

D. Erasing data (Lecture: 45; Lab: 20)

- (a) Introduction
- (b) Erasing Memory
    - i. Solutions
- (c) Erasing Files
    - i. Examples
    - ii. Solutions
- (d) Backups
- (e) Application/System Design issues
- (f) Summary
- (g) Lab

E. Race conditions (Lecture: 25; Lab: 10)

- (a) Introduction
- (b) TOCTTOU race conditions
    - i. passwd command races
    - ii. Temporary Files
    - iii. Avoiding TOCTTOU problems
- (c) Memory corruption Race Conditions
    - i. Multithreaded Processes
    - ii. Signal race conditions
    - iii. OS Kernel race conditions
- (d) Summary
- (e) Race Conditions Lab

F. Cryptography Fundamentals (Lecture: 30; Lab: 25)

- (a) Introduction
    - i. Cryptographic Applications
    - ii. Open design
- (b) Limits of Cryptography
- (c) Cryptographic Primitives
    - i. Cryptographic Hash Functions

    - ii. Symmetric key encryption
    - iii. Public key encryption
  - (d) Digital signatures
  - (e) Random Numbers
  - (f) Parameter sizes
  - (g) Insecure Cryptography
  - (h) Do Not Innovate in Cryptography!
  - (i) Summary
  - (j) Lab
- G. Using Cryptography (Lecture: 30; Lab: 30)
  - (a) Introduction
  - (b) Public Key Management
  - (c) Certificates
  - (d) Trust Models
  - (e) Example: PGP/GnuPG
  - (f) Example: SSL and TLS
  - (g) Using cryptography to improve security
  - (h) Standard strategies
  - (i) Standard Implementations
  - (j) Summary
  - (k) Lab
- H. Authentication (Lecture: 30; Lab: 35)
  - (a) The Basics
  - (b) Authentication using secrets
    - i. The problem with passwords
    - ii. Authentication protocols for secrets
  - (c) Authentication using physical tokens
    - i. Smart cards
    - ii. SecureID cards
  - (d) Biometrics
  - (e) Single sign-on
  - (f) US Authentication Standards
  - (g) Best Practices from OWASP
  - (h) Lab
- I. Research directions (Lecture: 20; Lab: 0)
  - (a) Introduction
  - (b) Static Analysis
  - (c) Dynamic Analysis
    - i. Limitations of Signatures and Specifications
    - ii. Anomaly detection

(d) Mitigation
(e) The Security Arms Race
(f) Summary

J. Final lab (Lecture: 0; Lab: 120)

(a) Lab