Course Description
In this course the student will learn about some of the most advanced topics on Information Assurance. With the increase in computer related crimes, it is imperative to develop software and applications in a secure way. The objective of this course is to teach students the methodology to write secure code applying the Secure Software Engineering life Cycle. In addition, students will learn the most important vulnerabilities in the C++ programming language.

Students Learning Outcomes:
After completing this course, the student should be able to:
Have a higher-level understanding of how to write secure code by using the Secure Software Engineering Life Cycle.
1. Comprehend, apply, and implement Secure Software Requirements
2. Comprehend, apply, and implement Secure Software Design
3. Comprehend and apply Secure Software Implementation
4. Comprehend, apply, and implement Secure Software Testing
5. Comprehend, apply, and implement Vulnerabilities in C++

Graded Activity
Assessment Mechanism Percentage
• Team Programming Assignment: Detection of vulnerabilities in C++ 40%
• Documentation for the Programming Project 20%
• Team paper on Secure Programming 30%
• Class participation 10%

Grading Scale
Grade   Scale
A  90-100%
B  80-89%
C  70-79%
D  60-69%
F  0-59%

Grading Notes and Comments
Incompletes: An incomplete will only be granted in the case of serious illness. Written proof of the illness and a recommendation for an incomplete will be required from both the Dean of Students office as well as from a doctor. A grade of incomplete is never issued to give a student more time to complete
assignments or improve a grade. The final determination as to whether or not an incomplete should be issued rests solely with the professor. Note: An 89 is a B

*Notice to Students with Disabilities: Texas A&M University-Corpus Christi complies with the Americans with Disabilities Act in making reasonable accommodations for qualified students with disabilities. If you suspect that you may have a disability (physical impairment, learning disability, psychiatric disability, etc.), please contact the Services for Students with Disabilities Office, located in Driftwood 101, at 825-5816. If you need disability accommodations in this class, please see me as soon as possible.

**ACADEMIC ADVISING: The College of Science and Technology requires that students meet with an Academic Advisor as soon as they are ready to declare a major. The Academic Advisor will set up a degree plan, which must be signed by the student, a faculty mentor, and the department chair. The College's Academic Advising Center is located in Driftwood 203E, and can be reached at 825-3466.

*** Grade Appeal Process. As stated in University Rule 13.02.99.C2, Student Grade Appeals, a student who believes that he or she has not been held to appropriate academic standards as outlined in the class syllabus, equitable evaluation procedures, or appropriate grading, may appeal the final grade given in the course. The burden of proof is upon the student to demonstrate the appropriateness of the appeal. A student with a complaint about a grade is encouraged to first discuss the matter with the instructor. For complete details, including the responsibilities of the parties involved in the process and the number of days allowed for completing the steps in the process, see University Rule 13.02.99.C2, Student Grade Appeals, and University Procedure 13.02.99.C2.01, Student Grade Appeal Procedures. These documents are accessible through the University Rules Web site at http://www.tamucc.edu/provost/university_rules/index.html. For assistance and/or guidance in the grade appeal process, students may contact the Office of Student Affairs.

Optional Texts
1. Secure Coding in C and C++ Robert C. Seacord

COURSE OUTLINE

First part: Secure Software Engineering

Why Is Security a Software Issue?

1.1 Introduction
1.2 The Problem
1.2.1 System Complexity: The Context within Which Software Lives
1.3 Software Assurance and Software Security
1.3.1 The Role of Processes and Practices in Software Security
1.4 Threats to Software Security
1.5 Sources of Software Insecurity
1.6 The Benefits of Detecting Software Security Defects Early
1.6.1 Making the Business Case for Software Security: Current State
1.7 Managing Secure Software Development
1.7.1 Which Security Strategy Questions Should I Ask?
1.7.2 A Risk Management Framework for Software Security
1.7.3 Software Security Practices in the Development Life Cycle

What Makes Software Secure?

2.1 Introduction
2.2 Defining Properties of Secure Software
2.2.1 Core Properties of Secure Software
2.2.2 Influential Properties of Secure Software
2.3 How to Influence the Security Properties of Software
2.3.1 The Defensive Perspective
2.3.2 The Attacker’s Perspective
2.4 How to Assert and Specify Desired Security Properties
2.4.1 Building a Security Assurance Case
2.4.2 A Security Assurance Case Example
2.4.3 Incorporating Assurance Cases into the SDLC
2.4.4 Related Security Assurance and Compliance Efforts
2.4.5 Maintaining and Benefitting from Assurance Cases

Requirements Engineering for Secure Software

3.1 Introduction
3.1.1 The Importance of Requirements Engineering
3.1.2 Quality Requirements
3.1.3 Security Requirements Engineering
3.2 Misuse and Abuse Cases
3.2.1 Security Is Not a Set of Features
3.2.2 Thinking About What You Can’t Do
3.2.3 Creating Useful Misuse Cases
3.2.4 An Abuse Case Example
3.3 The SQUARE Process Model
3.3.1 A Brief Description of SQUARE
3.3.2 Tools
3.3.3 Expected Results
3.4 SQUARE Sample Outputs
3.4.1 Output from SQUARE Steps
3.4.2 SQUARE Final Results
3.5 Requirements Elicitation
3.5.1 Overview of Several Elicitation Methods
3.5.2 Elicitation Evaluation Criteria
3.6 Requirements Prioritization
3.6.1 Identify Candidate Prioritization Methods
3.6.2 Prioritization Technique Comparison
3.6.3 Recommendations for Requirements Prioritization

Secure Software Architecture and Design

4.1 Introduction
4.1.1 The Critical Role of Architecture and Design
4.1.2 Issues and Challenges
4.2.1 Software Characterization
4.2.2 Threat Analysis
4.2.3 Architectural Vulnerability Assessment
4.2.4 Risk Likelihood Determination
4.2.5 Risk Impact Determination
4.2.6 Risk Mitigation Planning
4.2.7 Recapping Architectural Risk Analysis
4.3.1 Security Principles
4.3.2 Security Guidelines
4.3.3 Attack Patterns

Considerations for Secure Coding and Testing

5.1 Introduction
5.2 Code Analysis
5.2.1 Common Software Code Vulnerabilities
5.2.2 Source Code Review
5.3 Coding Practices
5.3.1 Sources of Additional Information on Secure Coding
5.4 Software Security Testing
5.4.1 Contrasting Software Testing and Software Security Testing
5.4.2 Functional Testing
5.4.3 Risk-Based Testing
5.5 Security Testing Considerations Throughout the SDLC
5.5.1 Unit Testing
5.5.2 Testing Libraries and Executable Files
5.5.3 Integration Testing
5.5.4 System Testing
5.5.5 Sources of Additional Information on Software Security Testing

Security and Complexity: System Assembly Challenges

6.1 Introduction
6.2 Security Failures
6.2.1 Categories of Errors
6.2.2 Attacker Behavior
6.3 Functional and Attacker Perspectives for Security Analysis: Two Examples
6.3.1 Web Services: Functional Perspective
6.3.2 Web Services: Attacker’s Perspective
6.3.3 Identity Management: Functional Perspective
6.3.4 Identity Management: Attacker’s Perspective
6.3.5 Identity Management and Software Development
6.4 System Complexity Drivers and Security
6.4.1 Wider Spectrum of Failures
6.4.2 Incremental and Evolutionary Development
6.4.3 Conflicting or Changing Goals Complexity
6.5 Deep Technical Problem Complexity

Governance, and Managing for More Secure Software

7.1 Introduction
7.2 Governance and Security
7.2.1 Definitions of Security Governance
7.2.2 Characteristics of Effective Security Governance and Management
7.3 Adopting an Enterprise Software Security Framework
7.3.1 Common Pitfalls
7.3.2 Framing the Solution
7.3.3 Define a Roadmap
7.4 How Much Security Is Enough?
7.4.1 Defining Adequate Security
7.4.2 A Risk Management Framework for Software Security
7.5 Security and Project Management
7.5.1 Project Scope
7.5.2 Project Plan
7.5.3 Resources
7.5.4 Estimating the Nature and Duration of Required Resources
7.5.5 Project and Product Risks
7.5.6 Measuring Software Security
7.6 Maturity of Practice
7.6.1 Protecting Information
7.6.2 Audit’s Role
7.6.3 Operational Resilience and Convergence
7.6.4 A Legal View
7.6.5 A Software Engineering View
7.6.6 Exemplars

Part 2: Secure Coding in C and C++

Running with Scissors.
- Gauging the Threat
- Security Concepts
- C and C++ Development Platforms

2. Strings.
- String Characteristics
- Common String Manipulation Errors
- String Vulnerabilities
Process Memory Organization
Stack Smashing
Code Injection
Arc Injection
Mitigation Strategies
Notable Vulnerabilities

3. Pointer Subterfuge.
   Data Locations
   Function Pointers
   Data Pointers
   Modifying the Instruction Pointer
   Global Offset Table
   The .dtors Section
   Virtual Pointers
   The atexit() and on_exit() Functions
   The longjmp() Function
   Exception Handling
   Mitigation Strategies

4. Dynamic Memory Management.
   Dynamic Memory Management
   Common Dynamic Memory Management Errors
   Doug Lea's Memory Allocator
   RtlHeap
   Mitigation Strategies
   Notable Vulnerabilities

5. Integer Security.
   Integers
   Integer Conversions
   Integer Error Conditions
   Integer Operations
   Vulnerabilities
   Nonexceptional Integer Logic Errors
   Mitigation Strategies
   Notable Vulnerabilities

6. Formatted Output.
   Variadic Functions
   Formatted Output Functions
   Exploiting Formatted Output Functions
   Stack Randomization
   Mitigation Strategies
   Notable Vulnerabilities

7. File I/O.
   Concurrency
   Time of Check, Time of Use
   Files as Locks and File Locking
   File System Exploits
Mitigation Strategies

8. Recommended Practices.
   Secure Software Development Principles
   Systems Quality Requirements Engineering
   Threat Modeling
   Use/Misuse Cases
   Architecture and Design
   Off-the-Shelf Software
   Compiler Checks
   Input Validation
   Data Sanitization
   Static Analysis
   Quality Assurance
   Memory Permissions
   Defense in Depth
   TSP-Secure