Miami Dade College

Course Description

CIS2619| Secure Software Development | 4.00 credits

This course provides an introduction to Secure Software Development in modern languages such as Java, C and C++. Common weaknesses exploited by attackers are discussed, as well as mitigation strategies to prevent those weaknesses. Students practice programming and analysis of software systems through testing and static analysis. Prerequisite: COP2800. Corequisite: COP2805C.

Course Competencies

Competency 1: The student will demonstrate an understanding of security principles by:

1. Restating the principles of confidentiality, integrity, availability, authenticity and non- repudiation
2. Describing what risk is in the context of software development and what constitutes an unacceptable risk
3. Explaining the meaning of essential security terminology such as vulnerability, hack value, exploit and payload.
4. Discussing the security, functionality and usability triangle
5. Describing of motives, goals and objectives of information security attacks
6. Using secure programming paradigms such as pair programming and code reviews
7. Describing why security has to be a consideration from the point of initial design and throughout the lifecycle of a product
8. Eliciting, analyzing and realizing security requirements

Competency 2: The student will demonstrate an understanding of secure coding practices in C/C++ by:

1. Describing common string manipulation errors, such as improperly bounded string copies, off-by- one errors, null termination errors, and string truncation
2. Writing programs that use mitigation techniques, such as input validation and string size checking, to deal with common string manipulation errors
3. Describing common integer errors, such as wraparound, conversion and truncation errors and non-, truncation errors and non-exceptional integer logic errors (wrong integer type, signed vs. truncation errors, and non-exceptional integer logic errors (wrong integer type, signed vs. exceptional integer logic errors (wrong integer type, vs. vs. exceptional integer logic errors (wrong integer type, signed vs unsigned, etc.)
4. Writing programs that use mitigation techniques, such as input validation, proper integer size selection, and use of abstract data types, to deal with common integer errors
5. Discussing the concept of information exposure through error messages
6. Explaining buffer overflows and how to prevent them
7. Writing programs that follow proper coding guidelines, such as using proper variable names, commenting, using code formatting conventions, etc
8. Explaining pointers and the security issues caused by their improper use

Competency 3: The student will demonstrate an understanding of secure coding practices in Java by:

1. Limiting the life of sensitive data in the program by restricting the scope of variables and objects
2. Avoiding duplication of code by proper use of methods and classes with specific functionality
3. Using encapsulation to hide information
4. Purging sensitive information from exceptions to avoid information exposure through error messages
5. Limiting the accessibility of classes, interfaces, methods, and fields
6. Limiting the extensibility of classes and methods
7. Explaining coupling and how to achieve loose coupling
8. Explaining cohesion and how to achieve high cohesion

9. Using the best practices to secure classes (data owned by a class should not be changed by reference, authentication, completion verification, context-based access control to the data owned by a class)
10. Discussing SQL injection attacks and how to avoid them
11. Explaining code injection attacks and how to prevent them
12. Using data obfuscation and data protection

Competency 4: The student will demonstrate an understanding of software testing techniques to identify security vulnerabilities by:

1. Explaining the role of testing in secure programming
2. Describing the different types of testing (e.g. black box, testing, white box testing, unit testing, system testing, acceptance testing) and their role in the software development life cycle
3. Using software testing terminology (e.g. test plan, test cases, actual results, expected results)
4. Creating and executing a test plan that covers all cases
5. Using an automated testing tool, such as JUnit
6. Using fuzzing to uncover vulnerabilities in a program
7. Discussing abuse cases and their use to uncover software vulnerabilities
8. Performing static analysis on a program to detect design errors and vulnerabilities
9. Explaining the concept of test-driven development

Competency 5: The student will demonstrate an understanding of third-party library security risks and mitigation strategies by:

1. Explaining the risks associated with the use of third-party libraries and components
2. Identifying all third-party libraries and versions in use in a program, including all dependencies
3. Monitor the security of third-party libraries and components in public databases, project mailing lists, and security mailing lists, and keep them up to date
4. Adding security wrappers (when appropriate) around third party components to disable unused functionality and/ or secure, weak, or vulnerable aspects of the component

Competency 6: The student will demonstrate an understanding of exception programming techniques and their importance in secure programming by:

1. Describing exceptions
2. Explaining the use of exceptions in the design, testing, and writing of secure and robust programs
3. Encapsulating exceptions
4. Throwing and catching exceptions

Competency 7: The student will demonstrate an understanding of debugging techniques using an Integrated Development Environment (IDE) and their use in secure programming by:

1. Downloading and installing an IDE
2. Understating all the functionality available Inan IDE
3. Designing, writing, and running programs in an IDE
4. Using the debugger to find programming errors (debugging)
5. Using common debugging functionality, such as Step Over, Step Into, Step Out, Run to cursor, etc
6. Using breakpoints to stop the execution of a program at a particular point of the execution
7. Using watches to monitor the value of variables during the execution of the program

<u>Learning Outcomes:</u>

- Use quantitative analytical skills to evaluate and process numerical data
- Formulate strategies to locate, evaluate, and apply information.
- Use computer and emerging technologies effectively