CST807: SECURE SOFTWARE ENGINEERING

AFRICA CENTRE OF EXCELLENCE ON TECHNOLOGY ENHANCED LEARNING (ACETEL)

NATIONAL OPEN UNIVERSITY OF NIGERIA

Course Guide for CST807

Introduction

CST807 – Secure Software Engineering is a 2-credit unit. The course is an elective course in first semester. It will take you 15 weeks to complete the course. You are to spend 65 hours of study for a period of 13 weeks while the first week is for orientation and the last week is for end of semester examination.

You will receive the course material which you can read online or download and read off-line. The online course material is integrated in the Learning Management System (LMS). All activities in this course will be held in the LMS. All you need to know in this course is presented in the following sub-headings.

Course Competencies

By the end of this course, you will gain competency to:

- Assess Software Development Vulnerabilities

Course Objectives

The course objectives are to:

- Provide the student with a deep understanding of the intricacies of securing programming
- Enable students to assess vulnerabilities in programming languages.

To introduce students to the various software analysis and models

Working Through this Course

The course is divided into modules and units. The modules are derived from the course competencies and objectives. The competencies will guide you on the skills you will gain at the end of this course. So, as you work through the course, reflect on the competencies to ensure mastery. The units are components of the modules. Each unit is sub-divided into introduction, intended learning outcome(s), main content, self-assessment exercise(s), conclusion, summary, and further readings. The introduction introduces you to the unit topic. The intended learning outcome(s) is the central point which help to measure your achievement or success in the course. Therefore, study the intended learning outcome(s) before going to the main content and at the end of the unit, revisit the intended learning outcome(s) to check if you have achieved the

learning outcomes. Work through the unit again if you have not attained the stated learning outcomes.

The main content is the body of knowledge in the unit. Self-assessment exercises are embedded in the content which helps you to evaluate your mastery of the competencies. The conclusion gives you the takeaway while the summary is a brief of the knowledge presented in the unit. The final part is the further readings. This takes you to where you can read more on the knowledge or topic presented in the unit. The modules and units are presented as follows:

Module 1: Fundamental Components of Design Architecture
- Unit 1: Architecture Development and Style
- Unit 2: Technological Developments
- Unit 3: Performance Measure

Module 2: Instructional Set Architecture and Design
- Unit 1: Memory Location and Operations
- Unit 2: Addressing Modes
- Unit 3: Instruction Types

Module 3: Secure Component Design
- Unit 1: Processing Unit Design
- Unit 2: Memory System Design
- Unit 3: Input and Output Design

Module 4: Security Design Principles
- Unit 1: Principles of Secure Design
- Unit 2: Principles of Software Security
- Unit 3: Principles of Protection Mechanism
- Unit 4: Trusted Security Base

There are thirteen units in this course. Each unit represent a week of study.

Presentation Schedule

The weekly activities are presented in Table 1 while the required hours of study and the activities are presented in Table 2. This will guide your study time. You may spend more time in completing each module or unit.

Table I: Weekly Activities

Week	Activity
1	Orientation and course guide
2	Module 1 Unit 1
3	Module 1 Unit 2
4	Module 1 Unit 3
5	Module 2 Unit 1
6	Module 2 Unit 2
7	Module 2 Unit 3
8	Module 3 Unit 1
9	Module 3 Unit 2
10	Module 3 Unit 3
11	Module 4 Unit 1
12	Module 4 Unit 2
13	Module 4 Units 3 and 4
14	Revision and Response to Questionnaire
15	Examination

The activities in Table I include facilitation hours (synchronous and asynchronous), assignments, mini projects, and laboratory practical. How do you know the hours to spend on each? A guide is presented in Table 2.

Table 2: Required Minimum Hours of Study

S/N	Activity	Hour per Week	Hour per Semester
1	Synchronous Facilitation (Video Conferencing)	1	13
2	Asynchronous Facilitation (Read and respond to posts including facilitator's comment, self-study)	3	39
3	Assignments, mini-project, laboratory practical and portfolios	1	13
	Total	5	65

Assessment

Table 3 presents the mode you will be assessed.

Table 3: Assessment

S/N	Method of Assessment	Score (%)
1	Portfolios	10
2	Mini Projects with presentation	30
3	Assignments	20
4	Final Examination	40
Total		100

Portfolio

A portfolio has been created for you tagged "**My Portfolio**". With the use of Microsoft Word, state the knowledge you gained in every Module and in not more than three sentences explain how you were able to apply the knowledge to solve problems or challenges in your context or how you intend to apply the knowledge. Use this Table format:

Application of Knowledge Gained

Module	Topic	Knowledge Gained	Application of Knowledge Gained

You may be required to present your portfolio to a constituted panel.

Mini Projects with presentation

You are to work on the project according to specification. You may be required to defend your project. You will receive feedback on your project defence or after scoring. This project is different from your thesis.

Assignments

Take the assignment and click on the submission button to submit. The assignment will be scored, and you will receive a feedback.

Examination

Finally, the examination will help to test the cognitive domain. The test items will be mostly application, and evaluation test items that will lead to creation of new knowledge/idea.

How to get the Most from the Course

To get the most in this course, you:

- Need a personal laptop. The use of mobile phone only may not give you the desirable environment to work.
- Need regular and stable internet.
- Need to install the recommended software.
- Must work through the course step by step starting with the programme orientation.

- Must not plagiarise or impersonate. These are serious offences that could terminate your studentship. Plagiarism check will be used to run all your submissions.
- Must do all the assessments following given instructions.
- Must create time daily to attend to your study.

Facilitation

There will be two forms of facilitation – synchronous and asynchronous. The synchronous will be held through video conferencing according to weekly schedule. During the synchronous facilitation:

- There will be one hour of online real time contact per week making a total of 13 hours for thirteen weeks of study time.
- At the end of each video conferencing, the video will be uploaded for view at your pace.
- You are to read the course material and do other assignments as may be given before video conferencing time.
- The facilitator will concentrate on main themes.
- The facilitator will take you through the course guide in the first lecture at the start date of facilitation

For the asynchronous facilitation, your facilitator will:

- Present the theme for the week.
- Direct and summarise forum discussions.
- Coordinate activities in the platform.
- Score and grade activities when need be.
- Support you to learn. In this regard personal mails may be sent.
- Send you videos and audio lectures, and podcasts if need be.

Read all the comments and notes of your facilitator especially on your assignments, participate in forum discussions. This will give you opportunity to socialise with others in the course and build your skill for teamwork. You can raise any challenge encountered during your study. To gain the maximum benefit from course facilitation, prepare a list of questions before the synchronous session. You will learn a lot from participating actively in the discussions.

Finally, respond to the questionnaire. This will help ACETEL to know your areas of challenges and how to improve on them for the review of the course materials and lectures.

Learner Support

You will receive the following support:

- Technical Support: There will be contact number(s), email address and chatbot on the Learning Management System where you can chat or send message to get assistance and guidance any time during the course.

- 24/7 communication: You can send personal mail to your facilitator and the centre at any time of the day. You will receive answer to you mails within 24 hours. There is also opportunity for personal or group chats at any time of the day with those that are online.

- You will receive guidance and feedback on your assessments, academic progress, and receive help to resolve challenges facing your stuides.

Course Information

Course Code:	CST 807
Course Title:	Secure Software Engineering
Credit Unit:	2
Course Status:	Elective
Course Blurb:	This introduces the learners to security requirements; specification of security requirements; software development lifecycle and security development lifecycle; programming languages and type-safe languages; best security programming practices; writing secure distributed programs; secure software, risk analysis, threat modelling, deploying cryptographic algorithms, defensive coding, penetration testing, static analysis, and security assessment; security for web and mobile applications.
Course Duration:	13 Weeks
Required Hours for Study:	65

Course Team

Course Developer:	ACETEL
Course Writer:	Mustapha Aminu Bagiwa PhD and Donfack Kana PhD
Content Editor:	Ismaila Idris PhD
Instructional Designer:	Inegbedion Juliet .O. PhD
Learning Technologists:	
Copy Editor:	Mr Awe Olaniyan Joseph

Module 1: Fundamentals and Requirement Level Analysis

Introduction

In this module, you are going to learn the basic step for building security into a software development life cycle to produce secure software. The module is organised into four units as follows:

Unit 1: Overview of Secure Software Engineering
Unit 2: Software Security Life Cycle
Unit 3: Software Quality Attributes
Unit 4: Security Requirement Gathering Principles and Guidelines

Unit 1: Overview of Secure Software Engineering

Contents

1.0 Introduction

Software is the heart of the modern world; you cannot work with any modern devices without software. Software is used virtually in all aspect of human endeavour. This usage ranges personal use to companies and institution up to the governments of every country in the world. This unit

introduces the concept of software engineering, security challenges in software and way these challenges can be solved.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- describe the steps in software engineering and
- explain why security should be embedded in software development.

3.0 Main Content

3.1 Introduction to Software and Software Engineering Concept

Software is everywhere from our homes to the streets down to our working places. Its functionalities depend not only on the environment which it is developed but upon compounded software demanding information systems which are connected and use the Internet as the medium of their interaction and information transfer. It is used virtually in all facet of life ranging from national infrastructures, banks, telecommunications companies, hospitals, supermarkets, gas stations, voting infrastructures, airline, to academia. Other numerous institutions also depend on software to perform the basic activities in the areas of its usage. Therefore, the software is of utmost importance to the overall functionalities of every society.

In developing software, the user needs are analysed, and the application is designed, constructed, and tested to ensure that it satisfies the defined user needs through the use of software programming languages. Because of the importance of software, guidelines for developing software are needed to obtain economically software that is reliable and work efficiently. This is generally obtained through the application of engineering principles in software development. Software engineering is a detailed study of engineering to the design, development and maintenance of software. It ensures that the application is built consistently, correctly, on time and on budget and within requirements.

Proper and correct functionalities of software at all phases of technological advancement is not the only condition to put into consideration but also, the availability of the software everywhere and at the right time to achieve the digital advancements in life which everyone is getting familiar with. But, The rate at which sensitive information is revealed to the public

by organizations/institutions that aim at storing, processing, and transmitting information (gotten from, e.g. financial transactions, social networks etc.) through the Internet using their software-agnostic systems is alarming. This makes the software systems that handle these activities of storing, processing, and transmitting this sensitive information exposed to unauthorised and unintentional users. In a nutshell, software-agnostic systems and other software-enabled functionalities have provided more undefended and prevalent access to sensitive information—including personal identities—they use.
What is software reliability?

3.2 Security Problems in Software

As software is used virtually in all facets of life, so also, the problems associated with software are also available in virtually all facets of life. These problems are everywhere; they are available on all the devices that we use (e.g. our handheld devices and cars, hospital equipment and pacemakers, etc.) and not only on the traditional computers that we know. These problems can creep up the security and safety of any systems they operate on. Software with security problems is everywhere. Security problems like coding bugs, e.g. buffer overflow, design problems (such as shifting error handling) and unwanted users who can have unauthorised access to and also attack them with the aid of malicious code. They can find a middle ground in the systems by taking advantage of software weaknesses. The main victim of software security problems are the Internet-enabled software applications, but with the growing extensibility and complexity of Internet-enabled software applications, they make software security even more exciting.

The functionalities of most systems are solely dependent on software, which makes it an excellent focal point for unwanted users, whose purposes may be economical, unlawful, terrorist, oppositional, or malicious. So, this makes securing of software to be of utmost importance. The reason why unwanted users find it easy to aim at software is that it is fundamentally guaranteed that there are going to be the occurrence of known weaknesses in the software that have known attack methods. This can be misused to disrupt one or more of the software's security properties or to change the software security state into an insecure state.

What mechanisms do you think can be used in securing software?

3.3 Security Mechanisms

There are so many application security products available in the market these days that tend to provide a solution to the problem of insecure

software. But, these solutions offered by these products are not all they are cracked to be, because they may aid in demonstrating, detecting and describing security problems. But they do very little when it comes to solving the security problem.

The mechanisms used in securing software include:

a. Which security strategy questions should be asked?
b. Use of risk management process to describe software security.
c. Incorporating software security practices into the software development life cycle.

3.3.1 Which Security Strategy Questions Should be Asked?

An organisation/institution alone cannot provide all the protections and preventive security mechanisms that organization/institution needs. To achieve these, the organisation/institution must communicate and interact with appropriate organisation/institution stakeholders. This is very important because it will help in determining the threats that may attack the system, the threat acceptance and flexibility if the threat is recognised. The following are questions from the organisation/institution viewpoint, whose answers can help in understanding security risks in achieving project goals and objectives:

i. What value needs to be protected?
ii. Which assets need to be protected to tolerate this value? Why must the assets be protected, and what would happen if the assets are not protected?
iii. What are the possible opposing circumstances and consequences that must be prevented and managed, and at what cost? How much disturbance can be endured before actions are taken?
iv. How are residual risk determined and effectively managed?
v. How are the solutions to these questions effectively integrated into the implementation and enforcement of security plan and strategy?

Provision of answers to these questions can aid in determining how much, where, and how fast to invest in providing profiling solutions to software security risk. In contrary to this, the organization/institution will find it tough to express and provide an effective security strategy and, therefore, this may lead to the inability of the organisation/institution to successfully oversee and manage enterprise, information, and software security.

Why are software easily attacked?

3.3.2 Use of Risk Management Process to describe Software Security

An important aspect of any approach of ensuring adequate security of software is the characterisation and use of continuous risk management

process. Security risks such as risks found in the outputs and results produced by each software development life-cycle (SDLC) phase during assurance activities, risks introduced by insufficient processes, and personnel-related risks are all software security risk. These software security risks can be characterised using risk management framework (RMF). Example of RMF, as shown in Figure 1 below depicts five stages of how a business organisation can characterise security risks. These stages include:

i. Understanding business context
ii. Identify and link the business and technical risk
iii. SynthesiSe and rank the risks
iv. Define the risk mitigation strategy
v. Carryout fixes and validate

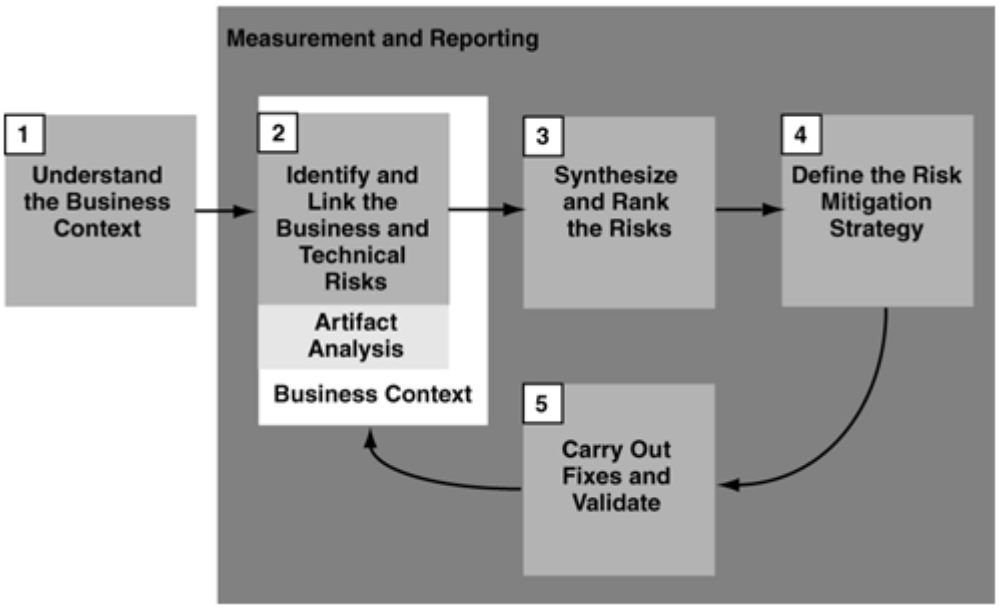

Fig. 1: Risk Management Framework

3.3.3 Incorporating Software Security Practices into the Software Development Life Cycle

The simple statement by software developers who focus on software functionality often misunderstood during the implementation of vital changes in the way software are built that; they take software security and security software to be the same. But, it is important to know that software security and security software are not the same during the implementation of vital changes in the way software are built. Clearly, there exist some security mechanisms, and most modern software includes these security mechanisms; but adding features such as SSL to any software (to protect communications cryptographically) does not provide a total solution to the security problem. Software security is a system-wide issue that takes take into consideration both design for security (such as a hard design that makes software attacks difficult) and security mechanisms. The main and reason why software security should

be incorporated into a full software development lifecycle approach are that the likelihood of security problem in any software is inevitable because of the presence of a problem in any system's standard-issue part than in some given security feature. Most methods used in practice these days in providing software security include training for developers, testers, and architects; analysis and auditing of software objects; and security engineering. But, a better way of providing software security will be to incorporate security practice in the software development life cycle. This can be applied irrespective of the base software development process being followed. In the end, this simply implied that a Secure Development Lifecycle (SDL) could be created by adjusting existing SDLC to incorporate the security practices into the development life cycle.

i. Which Security Strategy Questions Should be Asked?
ii. Use of Risk Management Process to describe Software Security
iii. Incorporating Software Security Practices into the Software Development Life Cycle

 **Discussion**

Let's reflect on the call for trustworthy computing by Bill Gates in 2002 on the future of software development.

Bill Gates: Priorities Are Changing, 2002

…In the past, we've made our software and services more compelling for users by adding new features and functionality, and by making our platform richly extensible. We've done a terrific job at that, but all those great features won't matter unless customers trust our software.

So now, when we face a choice between adding features and resolving security issues, we need to choose security. Our products should emphasize security right out of the box, and we must constantly refine and improve that security as threats evolve…Why is security not considered as part of the SDLC life cycle base on the lecture above?

4.0 Self-Assessment Exercise(s)

1. Software functionalities depend on
   I. Its development environment
   II. Internet for information transfer
   III. Compounded software demanding information systems which are connected
      A. I only
      B. I and II only
      C. II and III only
      D. I, II and III only

   ANSWER: D

Software functionalities depend on the environment in which it is developed, compounded software demanding information systems which are connected, and Internet, which serves as the medium of interaction and information transfer.

2. The following steps are required in software engineering
   I. Software design
   II. Software development
   III. Software consumption
   IV. Software reuse
   V. Software maintenance
      A. I and II only
      B. IV and V only
      C. I, II and V
      D. III, IV and V only
   ANSWER: C

Software engineering is a study of engineering to the design, development and maintenance of software

3. Why is Internet-based software the main victim of software security?
   A. It is designed by different software developers
   B. Unauthorised users can have access to the software
   C. Only authorised users can have access to the software
   ANSWER: B

Internet-based software is the main victim of software security because unauthorised users can have access to the software available online

4. Why is software security important?
   A. To prevent bugs and unwanted users
   B. To limit software usage
   C. To make software available to all intended users
   D. To provide proper and correct functionalities of the software
   ANSWER: A

5.0 Conclusion

As you have learnt in this unit, the software is very important in virtually all institutions. You have been taken through the concept of software engineering, the security problems in software and the mechanisms that can be used to solve these problems.

Assignment

On the 21st October 2019, Company-A requested a software product from Company-B to help handle their Customer financial data where customers

would input their personal details for registration and conduct transaction on the platform. Unfortunately, due to an input validation vulnerability in the software development process of Company-B Software product, some customers were able to View other customer account details stored in the database and manipulate their current balance using SQL injection. This stolen detail was used to clone credit card and make withdrawal as well as other transaction. From the information given above, write a comprehensive report on the precaution they should have taken to avoid these incidents.

6.0 Summary

Software is everywhere from our homes to the streets down to our working places. Its functionalities basically depend not only on the environment which it is developed but upon compounded software demanding information systems which are connected and use the Internet as the medium of their interaction and information transfer. It is used virtually in all facet of life. This facet of life includes national infrastructures, banks, telecommunications companies, hospitals, supermarkets, gas stations, voting infrastructures, airline, academia etc. Other numerous institutions also depend on software to perform the basic activities in the areas of its usage. But, this software can be threatened with security problems which can sabotage its importance. Security problems like bugs, flaws, viruses and unwanted intruders obtain unauthorised access to systems and also attack them with the aid of malicious code. They can find a middle ground in the systems by taking advantage of software weaknesses. Mechanisms like Which Security Strategy Questions Should be Asked?, Use of Risk Management Process to describe Software Security and Incorporating Software Security Practices into the Software Development Life Cycle are used in improving securing software.

7.0 References/Further Reading

Gary McGraw (2006). *Software Security: Building Security In*. Addison-Wesley. ISBN 0321356705.https://www.oreilly.com/library/ view/ software-security-building/0321356705/

Sommerville, I. (2011) *Software Engineering*. (9th ed.). Pearson.

Julia H. Allen; Sean Barnum; Robert J. Ellison; Gary McGraw & Nancy R. Mead (2006). *Software Security Engineering: A Guide for Project Managers*. Addison-Wesley. ISBN -10: 0-321-50917-X.

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead N. R. (2008). *Software Security Engineering.* Pearson India.

Mark S. Merkow & Lakshmikanth Raghavan (2010). *Secure and Resilient Software Development*. LLC. US: Taylor and Francis Group. https://doi.org/10.1201/EBK1439826966

Bill Gates: Trustworthy Computing https://www.wired.com/2002/01/bill-gates-trustworthy-computing/

Unit 2: Software security life cycle

Contents

1.0 Introduction

Software security is an important aspect of software engineering that tend to provide a solution to malicious attack onto the software. This unit explains the concept of the software development life cycle (SDLC) and also explains how software security can be incorporated in SDLC phases and its benefit.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- describe the phases of a software development life cycle
- explain how to build security in the SDLC

3.0 Main Content

3.1 The Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) is the standard process followed in developing any software product. It is an organized method of developing software applications. Most organisations adopt a model *(Waterfall, Iterative, Agile, etc.)* in developing software; this process may be modified according to the framework and requirement adopted by the organisation. When developing software, different standard SDLC models have been adopted in different ways to conform to individual conditions. This is explained in figure 2 below. The general SDLCs include the following phases as depicted in the figure that follows:

i. Requirements Gathering.
ii. Software Design.
iii. Coding.
iv. Testing.
v. Deployment.

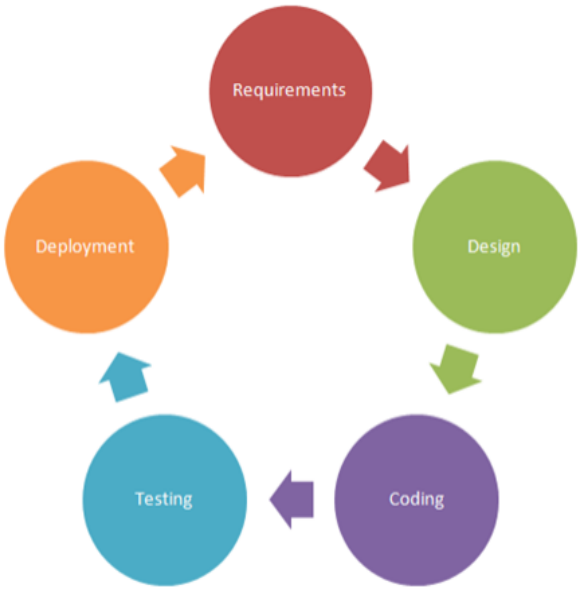

Fig. 2: The Software Development Life Cycle Phases

i. Requirements Gathering
This is the phase where expected behaviour of the software or system which needs to be developed are documented.

ii. Software Design
This is the phase where the scheme or template of the system which can be translated into software modules/functions/libraries, etc. is outlined. These pieces put together to form a software system. The output of this phase will be presented to the developer for proper automation via coding.

iii. Coding
In this phase, the scheme or template of the software is converted into reality by implementing the whole application using any programming language. Completion of the phase depends on the programming team involved (i.e. the number of people programming) and the size of the project (i.e. application).

iv. Testing
In this phase, testing of the project (i.e. application) is done. This is normally after the completion of the coding phase to ensure that the application performs expectedly and issues like performance and all functionalities are alright. If any problem concerning the performance and functionalities problem is observed, it will be fixed.

v. Deployment
After successful testing, and all performance and functionalities are alright. The application will be deployed into use in the phase base on purpose; it is meant to serve.

Most security issues emerge only after completion of the applications development. This can be taken care of by identifying the security issues through the security assessment performance of the applications. This is not the best way to treat security issues because the cost of implementation of this method is very high, and a large number of security issues will be discovered late (or not). So, a better way of taking care of security issues will be to incorporate the concept of security activities into the software development lifecycle. This can aid in discovering and reducing vulnerable activities.
What is the phase of SDLC?

3.2 Secured Software Development Life Cycle (SDLC)

This is simply the incorporating of security activities into the Software Development Life Cycle (SDLC) to ensure that security guarantee activities such as architecture analysis, review of code, and testing are an essential part of the development routine. Every phase of SDLC will include security – above and over the existing software development activities, as shown in the figure below.

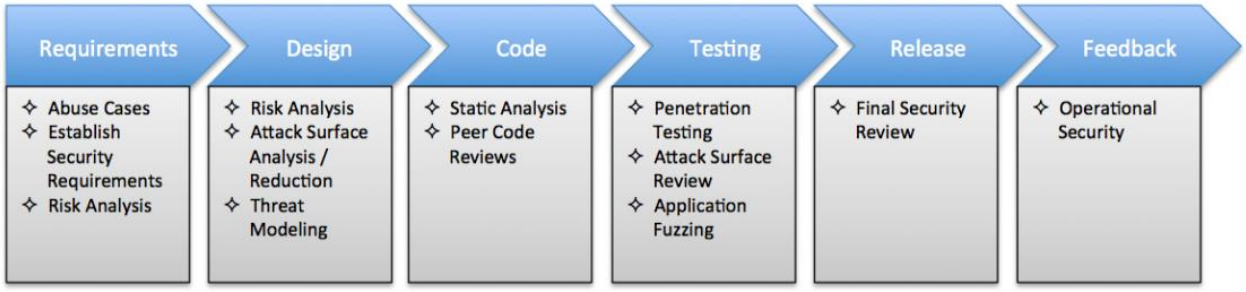

Fig. 3: The Secured Software Development Life Cycle Phases

The association of each software development lifecycle phase with corresponding security activities is shown in the figure below:

- Requirements
  - Abuse cases
  - Establish Security Requirements
  - Risk Assessment
- Design
  - Risk Assessment
  - Attack surface analysis/reduction
  - Threat Modeling
- Coding
  - Perform Static Analysis
  - Peer code Best Practices
- Testing
  - Penetration testing
  - Attack surface review
  - Application Fuzzing
- Deployment

- Final Security Review
- Feedback
  - Operational security

There are so many advantages in incorporating security into SDLC of any organisation's framework. These advantages include:

i. Development of secured software as security is a *persistent* problem.
ii. Makes stakeholders be security conscious.
iii. Early detection of security errors/fault/problems in the system.
iv. Cost of detecting and resolving security issues in the system will be reduced.

The restriction of the application of security controls to the requirements, design, code, and test phases in the software development life cycle is not proper. So it should be extended to deployment phase, where quality assurance, strict configuration control, security tests, and code reviews would be performed, together with system update because it adds effectiveness of secured software.

Virtual Laboratory Activity

Content Editor, please provide activity with solution for the students to work on

4.0 Self-Assessment Exercise(s)

1. The importance of incorporating security measure right from the SDLC are:
   i. Development of secured software as security is a *persistent* problem.
   ii. Makes stakeholders be security conscious.
   iii. Early detection of security errors/fault/problems in the system.
   iv. Cost of detecting and resolving security issues in the system will be reduced

   A. I only
   B. I, II and III only
   C. I, II, III and IV
   D. II and III

   ANSWER: C

2. Abuse cases are security activity in which software development lifecycle phase?
   A. Requirements Gathering.
   B. Software Design.
   C. Coding.
   D. Testing

   ANSWER: A

Requirements gathering is the phase where software developer will describe what the software will do when everything goes right.

3. The best way of treating software security is by identifying the security issue.
   A. True
   B. False

   ANSWER: B

Identifying the security issue is not the best way of treating software security because identifying software security issue is an expense. When treated, there is no guarantee that another security issue will not occur in the future.

Assignment
What is the security control that can be done at the requirement phase of SDCL?

i. quality assurance
ii. strict configuration control
iii. code reviews
iv. system update

5.0 Conclusion

In the unit, we discussed the phase of SDLC and stated the reason why security should be incorporated in all the phases of the SDLC, which will be cost-effective and more secured. We presented a secured SDLC, all the possible add security measures in each phase of the SDLC and also the benefits in a secured SDLC.

6.0 Summary

Software Development Life Cycle (SDLC) is the standard process followed in developing any software product. Most organisations adopt any type of SDLC model in developing software; this process may be modified according to the framework and requirement adopted by the organization. Most security issues emerge only after completion of applications. These security issues can be taken care of by identifying the security issues through the security assessment performance of the applications. This is not the best way to treat security issues because the cost of implementation of this method is very high, and a large number of security issues will be discovered late (or not). A better way of taking care of these security issues will be to incorporate the concept of security activities into the software development lifecycle. Incorporating security activities into SDLC of any organisation's framework can provide advantages like:

- the development of secured software as security is a *persistent* problem;
- stakeholders become security conscious;
- early detection of security errors/fault/problems in the system,
- and the cost of detecting and resolving security issues in the system are minimised.

7.0 References/Further Reading

Amjad H, Mohammad A. A, Ola M. S & Mohammed, A. (2017). "A Survey on Design Methods for Secure Software Development." *International Journal of Computer and Technology*. Vol. 16 Iss. 7https://rajpub.com/index.php/ijct/article/view/6467

Axelrod, C. W. (2013). *Engineering safe and secure software systems*. Artech House.https://us.artechhouse.com/Engineering-Safe-and-Secure-Software-Systems-P1556.aspx

Gary McGraw (2006). *Software Security: Building Security In*. Addison-Wesley. ISBN 0321356705.https://www.oreilly.com/library/view/software-security-building/0321356705/

Khan, M. U. A. & Zulkernine, M. (2009). "A Survey on Requirements and Design Methods for Secure Software Development," Technical Report No. 2009 – 562 , School of Computing, Queen's University, Kingston, Ontario, Canada, August 2009. http://research.cs.queensu.ca/TechReports/Reports/2009-562.pdf

Mark S. Merkow & Lakshmikanth Raghavan (2010). *Secure and Resilient Software Development.* LLC. US: Taylor and Francis Group, https://doi.org/10.1201/EBK1439826966.

Unit 3: Software Quality Attributes

Contents

1.0 Introduction

The previous unit introduced the concept of secured SDLC and its importance. In this unit, you will be taken through the attribute of quality software.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- describe the Attributes of good software
- investigate the quality of the software.

3.0 Main Content

3.1 Attributes of Software

The following attributes a good software must possess:

i. **Maintainability:** It should be dynamic in responding to users needs at all times. This is important because as the requirement and mode of operation of the environment change, functionalities of the software should also change.

ii. **Dependability and security:** It should be reliable, secure, and safe. It should not cause any damage (e.g. physical or economic) in the environment when a system failure occurs, and no unwanted users should have access or damage the system.

iii. **Efficiency:** It should properly manage the system resources such as memory and processor cycles.

iv. **Acceptability:** It should be acceptable to all the different users it is intended for, compatible with all other types of system, it must be usable.

What are the three principal dimensions to be achieved when securing any software?

3.2 Attributes of Secured Software

Several fundamental properties can be view as attributes of security as a software property; these properties include:

i. **Confidentiality:** The software must hide all the contents of the assets it is managing and the characteristics of the assets from unauthorised users or entities. Such characteristics as its relationships with its execution environment and its users. The software must also prevent unauthorised users and entities from having access to publicly available assets such as open-source software, its characteristics and content.

ii. **Integrity:** Unauthorised changes such as overwriting, corrupting, tampering, damaging, inserting of unintended (including malicious) logic, or deletion to software and its managed resources must be resistant and resilient to any mode of sabotage by unauthorised user and entities. This sabotage by unauthorised user and entities can be illegal changes to the software code, managed system resources, configuration, or behaviour by authorized entities, or any modifications by unauthorized entities. The integrity of software must be conserved at all level (that is during the software's development and its execution).

iii. **Availability:** At all time, the software must be available for use, functioning and in good condition to all its authorised and intended users (processes and human). At the same time, unauthorised users (humans and processes) should not be giving access to the functionalities of the software at all times also.

Two important additional properties of software entities (e.g., proxy agents, Web services, peer processes) that act as users:

i. **Accountability:** the software must record, track and acknowledge the responsibility of all software entities security-relevant actions. The security-relevant actions of the software entities should be specified by the audit-related language in the security policy of the software system. The tracking of the responsibility of all software entities security-relevant actions must be possible both before and after the recorded actions occur.

ii. **Non-repudiation:** the software must ensure that its accountability property cannot be undermined or avoided, and it must also have

the ability to prevent software entities from rejecting or challenging the responsibility of actions they have performed.

3.3 The Quality of a Software

Quality of software is not directly comparable with quality in manufacturing. Software quality is dependent on both functional and non-functional system attributes and not just about the correct implementation of the software functionality. In assessing the quality of software, there is a need to provide answers relating to the system's characteristics questions. Questions such as:

i. During the development process, are the programming and documentation standards followed?
ii. Is the testing of the software properly done?
iii. Can the software be sufficiently depended on when use?
iv. Is the software performance acceptable for normal use?
v. Can the software be used?
vi. Is the software understandable and well structured?

Attributes of quality of a software

Quality of software has attributes which are related to the software maintainability, dependability, efficiency and usability. These attributes are as follows:

i. Safety
ii. Understandability
iii. Portability
iv. Security
v. Testability
vi. Usability
vii. Reliability
viii. Adaptability
ix. Reusability
x. Resilience
xi. Modularity
xii. Efficiency
xiii. Robustness
xiv. Complexity
xv. Learnability

4.0 Self-Assessment Exercise(s)

1. The following are attributes of secured software except
   A. Confidentiality
   B. Maintainability
   C. Integrity
   D. Availability

ANSWER: B

2. All the following questions should be addressed when assessing the quality of software except
   A. Is the testing of the software properly done?
   B. Can the software be sufficiently depended on when use?
   C. Is the software performance acceptable for normal use?
   D. Is the software good?

   ANSWER: D

3. When software has the capability of producing a safety mechanism when ever it senses unwanted activity and also counters it if it finds a way into the software, we say the software is:
   A. Safe
   B. Secure
   C. Robust
   D. Good

   ANSWER: A

4. When software has the capabilities of distinguishing between any unwanted element that is trying to have access into it, we say the software is:
   A. Safe
   B. Secure
   C. Robust
   D. Good

   ANSWER: B

5. When software has the capabilities of recovering to normal working condition if any abnormality happens, then the software is said to be:
   A. Safe
   B. Secure
   C. Robust
   D. Good

   ANSWER: C

5.0 Conclusion

In this unit, we presented the attributes of software, and also attributes that a secured software should have and quality of good software.

6.0 Summary

Good software should have the following attributes:
i. Maintainability
ii. Dependability and security
iii. Efficiency
iv. Acceptability

While a secured software should have the following properties:
i. Confidentiality
ii. Integrity
iii. Availability
iv. Accountability
v. Non-repudiation

7.0 References/Further Reading

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software security engineering*. Pearson India.

Axelrod, C. W. (2013). "Engineering safe and secure software systems." Artech House.https://us.artechhouse.com/Engineering-Safe-and-Secure-Software-Systems-P1556.aspx

Mark S. Merkow & Lakshmikanth Raghavan (2010). *Secure and Resilient Software Development.* LLC. US: Taylor and Francis Group, https://doi.org/10.1201/EBK1439826966

McGraw, G. (2006). Software Security: Building Security In. Addison-Wesley. ISBN 0321356705.https://www.oreilly.com/library/view/software-security-building/0321356705/

Unit 2: Security Requirement Gathering Principles and Guidelines

Contents

1.0 Introduction

Security requirements gathering is an important aspect of secured SDLC. In this unit, we'll present the secured software requirement gathering process and also a case study.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- explain the security requirement phase
- perform security requirements gathering and analysis
- describe the security requirement of SDLC.

3.0 Main Content

3.1 Security Requirement Phase

As we all know, that security requirement of any system is different from the functional requirement of a system. Security requirement of any system is the high-level organisational policy presented into detailed system specification requirements. It centres on the analysis of system resources and services to be protected and also the security threats from which the system resources and services need protection

Quality Properties of Requirements

i. **Design Independent**: It should be free decisions to be taken when designing and developing any system
ii. **Unambiguous**: Is should the same response to a question is asked over and over
iii. **Precise**: It should specify precisely the behaviour of the software, its required data set and also the specification of input needed and its expected outputs.
iv. **Understandable**: the user should be able to understand its working principle.
v. **Traceable**: Identify the document uses the software
vi. **Verifiable**: A requirement is verifiable if there is a quantifiable or observable effect of the software that is directly expressed by the requirement.
vii. **Prioritised**: It should be able to rank processes base on their level of importance.
viii. **Complete**– It should contain all the information that describes the design pattern of the software so that all expect activities of the software and all outputs are presented. So also a response to user input should be documented, and the identification of all content of the system should be presented.
ix. **Consistent**: It should contain a statement that is precise in terms of meaning and the activity to be performed.
x. **Organised**: It should be well arranged so that a user can understand and identify the meaning and use of every statement.
xi. **Modifiable**: It can be modified/changed when the need arises.

3.2 Requirement Engineering Process

i. **Requirements Elicitation:** This is the process of observing the existing system and developing a prototype that can serve as a sample. This activity of observing the existing system may be by asking the users its working principle, shortcomings, and how they think it can be improved.
ii. **Requirements Analysis:** This is the evaluation of the conditions of the existing system to access the reason that preempted the development of the new system. This can be achieved by studying the results obtained from the requirements elicitation phase. The security requirements to put into consideration in this phase are:
    a. Confidentiality
    b. Integrity
    c. Availability
    d. accountability and
    e. authenticity.
iii. **Requirements Modelling and Specification:** In this phase, a document containing the information that translates the findings

from the requirement analysis phase will be developed to specify the modelling requirements.

iv. **Requirements Verification:** In this phase, all activities in the previous phase are checked to verify their correctness and also eliminate errors if they exist.

3.2.1 Secured Requirement Gathering Process

The software requirement gathering for secured SDLC consists of three activities, as shown in the diagram (Secure Software Development Life Cycle Phases) in Unit 3. These activities include:

i. Misuse/abuse case
ii. Establishing a security requirement
iii. Risk analysis

3.3 Case Study: Banking System

Consider applying these secured software requirement gathering on the banking system.

i. Misuse/abuse case

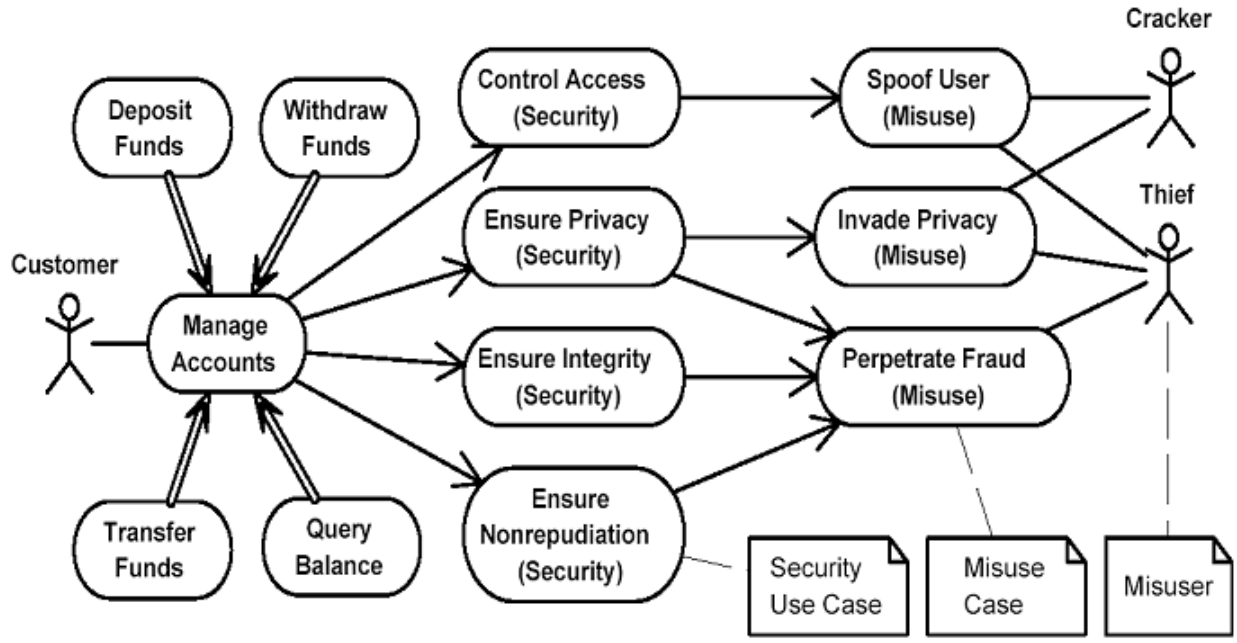

Fig. 4: Security Requirement

As shown in the Figure above, the misuse cases are incorporated into use case diagrams to express the system unwanted behaviours (e.g., spoofing user account, invading privacy and perpetrate fraud) instigated by a misuser (e.g., cracker or thief). This depiction results in security use cases are controlling access, ensuring privacy, integrity and nonrepudiation.

ii. Establishing a security requirement

When establishing a security requirement gathering in the baking system, take the following steps:

a. System modelling.
b. Identification of resources.
c. Identification of threats and vulnerabilities.
d. Elicitation of security requirements.
e. Evaluation of security requirements.

a. **System modelling:** this models the baking system into subproblems of bank, bank staff, bank customer and account information.
b. **Identification of resources:** This is the checking of subproblems the system modelling can create. For example, information on any customer account can be created into two subproblems:
   i. editing of account information and
   ii. viewing account information.
c. **Identification of threats and vulnerabilities**: In this step, identify possible threats that will be harmful to the system by exploiting the system vulnerabilities. Will the threats describe the capabilities of the attacker to violate the security concerns of the system?
d. **Elicitation of Security requirements**: In this step, the modelling of intended security requirements to mitigate the threats causing vulnerabilities will be done.
e. **Evaluation of Security requirements**: this is the itemisation of the activities to de done to evaluate the resulting security requirements.

4.0 Self-Assessment Exercise(s)

i. The following are possible email misuse cases except.
   a) Eavesdropping on e-mail
   b) Attacks against the mail servers
   c) Modifying e-mail
   d) Communicate with a colleague
   e) Spoofing e-mail

   Answer: D

ii. Sending mail to a wrong recipient can be regarded as email misuse? TRUE or FALSE
   Answer: TRUE

5.0 Conclusion

Security requirement of any system is the high-level organisational policy presented into detailed system specification requirements. The software requirement gathering for secured SDLC consists of three activities:

i. Misuse/abuse case
ii. Establishing a security requirement
iii. Risk analysis

6.0 Summary

In the unit, we presented the quality properties of requirements gathering and requirement engineering process. We also presented a security requirement case study of a banking system.

7.0 References/Further Reading

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software security engineering*. Pearson India.

Amjad H, Mohammad A. A, Ola M. S & Mohammed, A. (2017). "A Survey on Design Methods for Secure Software Development." *International Journal of Computer and Technology.* Vol. 16 Iss. 7

Axelrod, C. W. (2013). *Engineering safe and secure software systems*. Artech House. https://us.artechhouse.com/Engineering-Safe-and-Secure-Software-Systems-P1556.aspx

Mark S. Merkow & Lakshmikanth Raghavan (2010). "Secure and Resilient Software Development." LLC. US: Taylor and Francis Group, https://doi.org/10.1201/EBK1439826966

McGraw, G. (2006). *Software Security: Building Security In*. Addison-Wesley. ISBN 0321356705.https://www.oreilly.com/library/view/software-security-building/0321356705/

Noopur, D. (2005), "Secure Software Development Life Cycle Processes: A Technology Scouting Report", *Software Engineering Process Management.*

Ransome, J., & Misra, A. (2018). *Core Software Security: Security at the Source*. CRC presshttps://www.crcpress.com/Core-Software-Security-Security-at-the-Source/Ransome-Misra/p/book/9781466560956

Module 2: Vulnerabilities during Implementation, Consequences, and Prevention, Consideration

Module Introduction

Software security is a branch in software engineering focusing on safe software design which involves the best use of programming language, tools and methods. In other words, secure software focuses on preventing vulnerabilities, faults and bugs in software. This module introduces you to techniques and methods for eliminating software vulnerabilities when building secured software. In this module, you will be studying six units as follows:

Unit 1: Defensive Coding Practices
Unit 2: Code Inspections
Unit 3: Database Security
Unit 4: Software Vulnerabilities And Exploitation
Unit 5: Secure Programming for Preventing BOF, FSB, SQLI, XSS, Session
Unit 6: Mobile Application Development Security

In each unit, I will explore a particular topic in detail and highlight self-assessment exercises at the end of the unit. Finally, I also highlight resources for further reading at the end of each unit.

Unit 1: Defensive Coding Practices

Contents

1.0 Introduction

Defensive coding is not referring to the act guarding your program codes. In defensive coding, a software developer is expected to exhibit a mind-set that he is not certain about what a cyber-criminal will do to attack his program. So that in case a malicious activity is lunched on his program, it will not have a negative effect on the operation of his software. In defensive programming, therefore, the concept is that if a method in a program passed a corrupt data, the data should not affect the functionality of the method even if the passed data is coming from another method or subroutine. In this unit, you will be equipped with skills on how to develop programs that will not be affected by invalid data, events or programmer mistake.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- describe the process of code inspection for errors
- develop projects using techniques of defensive programming.

3.0 Main Content

3.1 Guard Your Program to Isolate the Problem Caused by Errors

Barricade is a strategy for damage-containment. This is similar to having isolated compartments in the hull of a ship. If the vessel enters an iceberg and the hall is broken wide open, the compartment affected will be closed off, and the remainder of the vessel will not be impacted. Barricade is also comparable to a building firewall as well. A building's firewall prevents fire spreading from one part of a building to another. (Barricades used to be called' firewalls,' but the word ' firewall' now frequently relates to preventing hostile network traffic.) For defensive coding reasons, one way to barricade is to identify certain interfaces as ‘secure' areas.

3.2 How Much Defense to Leave in Production Code

The goal of defensive coding is the ability to have an error noticed as early at the development stage of software. This is because the developers need to be cautious at that stage because, during development, it is less expensive to fix such errors. Moreover, the program can always fail and recover gracefully during development without the loss of sensitive information. I will now introduce you to some guidelines on the programming code to leave and leave out in your production code during software development.

Why is it important to identify errors at an early stage of software development?

3.2.1 Allow Codes for Error Checking in the Program

An initial task is to decide on places in your program where errors can be catastrophic or non-catastrophic. Example, if you are developing a cashbook ledger system using a spreadsheet, an error for a screen update may not have severe consequences on the system because it will only affect the screen. However, you certainly cannot afford to have an error in the logical or arithmetical engine of the system. This sought of error will result in the production of the incorrect result by anyone using the system. Users will be more comfortable dealing with bad screen update rather than incorrect ledger calculations.

3.2.2 Do Away with Codes That Check for Trivial Errors

If an error cannot be fixed easily and present itself as non-trivial in term of its consequence, then remove the code that checks for it in the program. Code removal, in this case, is not referring to physical deletion. Instead, it refers to the use of version control, pre-compiler switches or any other user-defined technique that will allow the program to compile leaving out that code with an error. However, you can also leave the code that checks the error, but the error should always be recorded in an error log file whenever the program compile.

3.2.3 Remove Code That Results in Hard Crashes

As discussed in section 3.1, you need errors to be noticed as early as possible during the stages of your software development. This will allow you to fix that error without causing any damage since the system has not gone into full operation. Therefore, the best way to achieve this early fix is to have the program keep track and print debugging messages and crashes when an error is detected.

3.2.4 Leave in Code that Helps the Program Crash Gracefully

Allow programs that contain debugging code for fatal error detection in your program. For instance, engineers left some of their debug code in by design in the Mars pathfinder. An error occurred after the landing of the pathfinder. By using the debug code left in the complete program, JPL technician could diagnose the issue and upload an updated code version to the pathfinder.

3.2.5 Log Errors for Your Technical Support Personnel

Software developers usually equip their codes with checkpoint and assertions to halt a program for debugging purpose during program development. You might consider changing the checkpoint and assertions to message logs that will be saved in a file to be used by technical staff for clues in filing errors rather than physically deleting them.

3.2.6 Make Sure the Error Messages You Leave in Are Friendly

If you decide to practice the skills of leaving error messages in your program, make sure they are user friendly in terms of language and ease of understanding.

3.3 Defensive about Defensive Coding

Avoid too much defensive programming. This is because when you try to check and analyses every data that come into your program for every possible error, whether logical or semantic, your program will end of becoming so heavy and extremely slow.

The complexity of the program will be much higher because of the additional code needed for the defensive coding. The codes used for the defence may also contain defects. Thus, you may likely find a defect in the defensive system as well.

 **Discussion**

The use of healthy defensive programming in software development to avoid error after deployment is very important. Does defensive coding affect the complexity of a program, and what are your thoughts?

4.0 Self-Assessment Exercise(s)

1. Why do software developers usually equip their codes with checkpoint and assertions?
   a. For easy identification
   b. For easy lookup
   c. For easy debugging
   d. For easy execution

   Answer: C

Software developers usually equip their codes with checkpoint and assertions to halt a program for debugging purpose during program development.

2. It is good to have too many defensive codes in your software. True or False

Answer: B

It is not always good to have too much defensive code in your software. This is because when you try to check and analyses every data that come into your program for every possible error, whether logical or semantic, your program will end up becoming so heavy and extremely slow.

3. It is possible to find a defect in defensive code as well. True or False

Answer: True
You may likely find a defect in the defensive code as well.

5.0 Conclusion

You have learnt from this unit the skills on how to develop programs that will not be affected by invalid data, events or programmer mistake. Furthermore, you have also learnt that it is not always the best to be over defensive in your program. Thus, be careful about where to be defensive and set priorities for your defence.

6.0 Summary

At the end of this unit, you have learnt that defensive coding is not referring to the act of guarding your program codes but rather an idea on

the way to stop a corrupt data from affecting the functionality of your program. Even if the passed data is coming from another method or subroutine. In the next unit, you will be learning about code inspection.

7.0 References/Further Reading

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software Security Engineering*. Pearson India.

Ammann, P., & Offutt, J. (2016). *Introduction to Software Testing*. Cambridge University Press.https://cs.gmu.edu/~offutt/softwaretest/

Axelrod, C. W. (2013). *Engineering Safe and Secure Software Systems*. Artech House.https://us.artechhouse.com/Engineering-Safe-and-Secure-Software-Systems-P1556.aspx

Fernandez, E. B. (2004, June). "A Methodology for Secure Software Design." *Software Engineering Research and Practice* (pp. 130-136). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.2972&rep=rep1&type=pdf

Howard, M., LeBlanc, D., & Viega, J. (2005). *19 Deadly Sins of Software Security. Programming Flaws and How to Fix Them*.http://math.uaa.alaska.edu/~afkjm/cs470/handouts/SecuritySins.pdf

McGraw, G. (2006). Software Security: Building Security. (Vol. 1). Addison-Wesley Professional.https://www.oreilly.com/library/view/software-security-building/0321356705/

Ransome, J., & Misra, A. (2018). *Core Software Security: Security at the Source.* CRC Press https://www.crcpress.com/Core-Software-Security-Security-at-the-Source/Ransome-Misra/p/book/9781466560956

Viega, J., & McGraw, G. (2011). "Building Secure Software: How to Avoid Security Problems the Right Way" (paperback)(Addison-Wesley Professional Computing Series). Addison-Wesley Professional. https://www.oreilly.com/library/view/building-secure-software/9780672334092/

Unit 2: Code Inspections

Contents

1.0 Introduction

Code inspection is an essential component of checking and validating software. Code inspection is a mechanism used to analyse and verify system requirements, design model, source codes of a program and proposed system tests. In this unit, you will be learning about software inspection how it is performed and its advantages in developing a secured software system.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- explain the code inspection
- evaluate software application to determine if it has met the coding standard
- clarify if the software meets a code inspection standard.

3.0 Main Content

3.1 What is Code Inspection?

Code inspection sometimes referred to a critical code review is a special kind of review or static testing that focus on the check and analysis of source code of a software system to avoid multiplication of defect at a later stage of development. Code inspection has been very economical and effective in the detection of software defects.

Michael Fagan first introduced code inspection in software development and was used by IBM for many years before the first publication in an academic journal. Inspection on the readable portion of software like requirements or design models can also be performed. Code inspection includes the understanding of the system, its application domain and modelling domain to identify errors.

A later study at IBM reported that software developers spent 3.5 hours finding a single error when not using code inspection methodology.

Who usually leads the team of code inspectors?

Code inspection is most often led by an expert who in most cases, is not the writer of the code to be inspected. It involves a formal type of review based on defined rules and checks that uses exit and entry criterion which involves the individual examination of entire or part of a code of a software application. Code inspection aims at covering four main objects during software production as:

i. Verification of the conformance of the code with the concerned project documentation such as software design documents that are needed to clearly explain data handling, algorithms and traceability with the definition of requirements that the software must conform.
ii. Verification of the correct design. And the implementation of barriers to protect against internal and external attacks.
iii. Verify proper usage of programming rules.
iv. Verify the correct implementation of safety requirements.

Code inspection can be performed using manual code review, or it can be performed using a static analysis tool. I will now introduce you to the two ways of performing code inspection.

3.1.1 Manual Code Review

This involves a team of experts coming together with the code author to manually inspect the entire or part of software code to discover defects. Note that it is more appealing to identify the defect before deploying a

system after its complete development. Depending on the coding language used for the software, manual code inspection might present itself as simple or complex.

3.1.2 Static Code Analysis

This involves the systematic analysis of software without the need to execute its code. Note that analysis performed when the software is executing is called dynamic code analysis. Static analysis is usually done using with an automated tool in conjunction with a human intervention called program understanding, program comprehension or code review.

What is the advantage of static code analysis?

Static code analysis has the advantage of being able to inspect an entire software code. However, it may not be as efficient as an expert when it concerns defect discovery. Many static analysis tools are available online. Some of which are open source while others are proprietary.

3.2 Purpose of Code Inspection

There are about three main reasons for performing code inspection during software development which includes:

- Code inspection, which is usually done during software development to find defects and identify process improvement in the development phase.

- Code inspection if done, will report a list of findings that include performance metrics that can be used to assist in improving the software development process.

- Code inspection involves reading and understanding a source code which may aid an inspection.

3.3 Code Inspection Methodology

Code inspection methodologies in software development include code review and code walkthrough.

3.3.1 Code Review

Code review is the systematic evaluation of software code with an attempt to finding and removing vulnerabilities in the program code. Example of such vulnerabilities includes memory leaks, or buffer overflows. Code review should be well documented and normally includes experts as such; it should be led by a trained expert who is not the author of the code.

3.3.2 Code Walkthrough

Code walkthrough is a form of code inspection in which a programmer led the inspection and other team members participate by asking questions and cheeking and identifying errors about development standards. The designer usually is part of the meeting with some of his team members in attendance. The goal of this inspection methodology is to enhance learning about the content of the code. And also to find defects.

 **Discussion**

Does code inspection take care of SDLC in terms of secure coding?

Assignment

After reading this unit, what do you think are the responsibilities of a code reviewer? Start your response by explaining the code review and then stating your answer with theories and skill from your experience.

4.0 Self-Assessment Exercise(s)

1. What is the name of the person that introduced code inspection in software development?
   a. Ian Summerville
   b. Michael Farad
   c. Michael Hudson
   d. Michael Fagan

   Answer: D
   Michael Fagan

2. How long does a software developer spend on an average finding a single error in his program when not using code inspection?
   a. 2.5 hours
   b. 3.5 hours
   c. 4.5 hours
   d. 6.5 hours

   Answer: B
   3.5 hours

3. In how many ways can code inspection be performed?
   a. Two ways
   b. Three ways
   c. Four ways
   d. Five ways

Answer: A
Code inspection can be performed in two ways: manual code review and using static analysis tools.

5.0 Conclusion

You have learnt from this unit different ways in which inspection can be done on part or the entire part of the software code to identify defects. You have also learnt the objectives and purpose for code inspection in the process of software testing. In conclusion, you have ascertained that code inspection is an integral part of software verification and validation.

6.0 Summary

At the end of this unit, you have learnt that code inspection is critical for quality check and defect discovery in software development. You have also learnt that code inspection an essential step in the development of defect-free and robust software application. In the next unit, you will be learning about security in databases.

7.0 References/Further Reading

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software Security Engineering*. Pearson India.

Ammann, P., & Offutt, J. (2016). *Introduction to Software Testing*. Cambridge University Press.https://cs.gmu.edu/~offutt/softwaretest/

Fernandez, E. B. (2004, June). "A Methodology for Secure Software Design." In: *Software Engineering Research and Practice* (pp. 130-136). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.2972&rep=rep1&type=pdf

McGraw, G. (2006). *Software Security: Building Security* In (Vol. 1). Addison-Wesley Professional.https://www.oreilly.com/library/view/software-security-building/0321356705/

Viega, J., & McGraw, G. (2011). *Building Secure Software: How to Avoid Security Problems the Right Way* (paperback)(Addison-Wesley Professional Computing Series). Addison-Wesley Professional. https://www.oreilly.com/library/view/building-secure-software/9780672334092/

Unit 3: Database Security

Contents

1.0 Introduction

You will learn from this unit the approach that can be employed to secure your database against threats. You will also learn the mechanism for granting access privileges to the user and database security measures. You will also learn the techniques for enforcing security on a database.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- describe Data Security Triad
- explain Access control as it relates to database security
- utilise security control techniques to protect your database and data.

3.0 Main Content

3.1 Introduction to Database Security

Database security is a vast area in the design and development of databases stressing some concerns that include:

- Legal and ethical issues with regards to access rights to information. For example, some information out there may not be publicly accessible to unauthorized organisation or person because the information is private. In the digital world, several laws guide information privacy.

- The concern for policy issues either in government or cooperation institutions in terms of which type of data to be allowed public access. Examples are credit card security details or patient medical records.

- The concern of which level to secure the database system. An example is whether security should be implemented at the physical, operating system or the database management system level.

- The concern for an organisation to identify and categorise user and data at different security levels. An example is credential access level such as high, medium or level access.

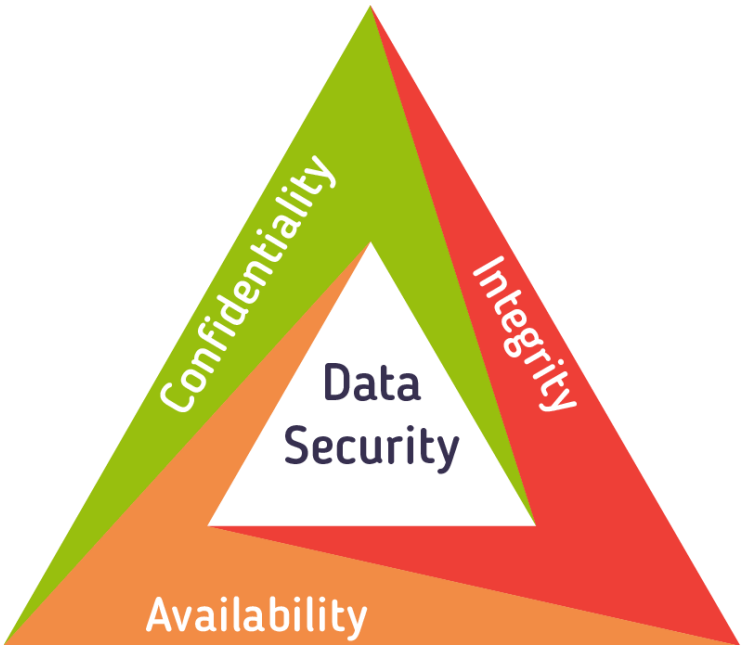

Fig. 5: Data Security Triad

Checkmarx.com

3.2 Database Security Threats

When cyber-criminal lunch an attack on the database of organisation, the organisation may end of suffering from degradation or loss data and privacy and security goals such as integrity, confidentiality and availability as seen in figure 5.

3.2.1 Loss of Integrity

Database integrity is a necessity for the protection of data from an illegal alteration in a database. Alteration of information involves creating, insertion, update, changing information status and deleting of information.

3.2.2 Loss of Availability

Data integrity loss happens in a database if changes to the information in the database are made intentionally or unintentionally without authorized permission. If the integrity of information is breached, it may lead to erroneous information.

3.2.3 Loss of Confidentiality

Database confidentiality is a requirement that the data in the database be protected against unauthorised public disclosure.

What is an example of confidentiality loss?

Example of confidentiality loss includes data privacy violations. Therefore, to protect databases against these security threats, I will introduce you to four kinds of control measures.

3.3 Database Security Control Measures

Four main control measures are available to provide security to databases. These control measures are:

i. Access control
ii. Inference control
iii. Flow control
iv. Encryption

3.3.1 Access control

One security problem surrounding all computer systems is authorised to access to data for malicious reasons. Therefore, DBMS have to include mechanisms for the restriction of illegal database access. Such a mechanism is referred to as access control and can be handled by the creation of different user accounts that could control logins by the DBMS.

3.3.2 Inference Control

Based on different criteria, statistical databases are used to provide statistical data or summaries of values. A population statistics database, for instance, can provide age-specific statistics, revenue level and other criteria. Access to the database may be allowed to statistical database users such as corporate statisticians or commercial businesses to acquire statistical data about a population but not to access extensive private data about individuals.

In-Text Question

What should be ensured by statistical database security?

Statistical database security must ensure that unauthorised people do not gain access to an individual's private information. Sometimes deducting or inferring from queries involving only summary statistics based on organisational information is possible. This must, therefore, not be allowed. This situation is called the security of the statistical database, and the associated control measures are called control measures for inferences.

3.3.3 Flow Control

Flow control is another security measure that is used in database security information flow prevention such that the information does not fall into the hands of an unintended user. Many channels serve as pathways for the flow of information that violates organization security policies, and such pathways are called covert channels.

3.4.4 Encryption

Data encryption is also another security measure for the protection of sensitive information (such as credit card numbers) that may be transmitted over the Internet. Encryption is also used in providing additional security for the protection of the sensitive part of a database. The data in the database usually are encoded using some encryption algorithm. An unauthorised user who may eventually succeed in accessing the encrypted data may not make sense of it easily, but authorised users are given decoding or decrypting algorithms (or keys) to decipher the data. Encrypting techniques are very difficult to decode without a key.

3.4 Database Administrator and Security

The Database administrator (DBA) controls the activity of the database and managing its usage. He/she is responsible for granting access to users of the database and categorizing the users concerning the organisation security policies. The DBA also has a DBA account called a system or superuser account in the database management system that offers strong rights not made accessible to periodic database accounts

and users. DBA-privileged commands include instructions to grant and withdraw rights to individual accounts, users or groups of users.

 **Discussion**

After reading this unit, State four ways of providing security to your database. Start your response by explaining what database security means.

 **Case Studies**

On a regular checkup, Mr Aliyu engaged in a normal Google search, operating system fingerprinting and port scans. However, he could not gain access instead got an encrypted authentication login form using SSL. After a careful check on the webpage containing the encrypted login form, Mr Aliyu noticed that a hidden APP_Name field is being passed to his database application whenever there is an attempt to login to the site resulting in an injection to his database.

Virtual Laboratory Activity

Content Editor, please provid activity and solution on coding and data security

4.0 Self-Assessment Exercise(s)

1. What do organisations suffer from when cybercriminals launch an attack on their database?
   a. Data degradation only
   b. Loss of data only
   c. Degradation and loss of data
   d. None of the among the options

   Answer: C

The organisation usually suffers from degradation and loss of data when there is an attack on their database by cybercriminals. Because the integrity, confidentiality and availability of the data in the database would have been compromised.

2. What are covert channels?
   a. Covert channels are pathways for the flow of information that does not violate organisation security policies.

b. Covert channels are pathways for the flow of information that violates organisation security policies.
c. Covert channels are codes for the encrypting information
d. Covert channels are codes for the decrypting information

Answer: B

Covert channels are pathways for the flow of information that violates organization security policies.

3. Which among the option is not the responsibility of a database administrator (DBA)?
   a. Controlling the activities of the database
   b. Managing the usage of the database
   c. Granting access to users of the database
   d. Designing DBMS

   Answer: D

The database administrator (DBA) is responsible for controlling the activities of the database and managing its usage. He/she is responsible for granting access to users of the database and categorising the users concerning the organisation security policies. A DBA is not responsible for designing DBMS.

5.0 Conclusion

You have learnt from this unit the approach that can be employed to secure a database against threats. You have also learnt about some database security threats and how to enforce access control measures on a database. In conclusion, you have ascertained that the DBA is responsible for the overall security of the database system.

6.0 Summary

At the end of this unit, you have learnt the concept of database security and the different database threats in terms of loss of integrity, availability, and confidentiality. You have also learnt the types of control measures to deal with these problems, which include access control, inference control, flow control, and encryption. In the next unit, you will be learning about software vulnerabilities and how they are exploited by the cybercriminal to launch an attack to target machines.

7.0 References/Further Reading

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software Security Engineering.* Pearson India.

Ammann, P., & Offutt, J. (2016). *Introduction to software testing*. Cambridge University Press.https://cs.gmu.edu/~offutt/softwaretest/

Axelrod, C. W. (2013). *Engineering Safe and Secure Software Systems.* Artech House.https://us.artechhouse.com/Engineering-Safe-and-Secure-Software-Systems-P1556.aspx

Fernandez, E. B. (2004, June). "A Methodology for Secure Software Design." In: *Software Engineering Research and Practice* (pp. 130-136).http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.2972&rep=rep1&type=pdf

Howard, M., LeBlanc, D., & Viega, J. (2005). *19 Deadly Sins of Software Security. Programming Flaws and How to Fix Them*.http://math.uaa.alaska.edu/~afkjm/cs470/handouts/Security Sins.pdf

Ramez E & Navathe S (2011). *Fundamentals of Database*. (6th ed.). Addison-Wesley

Ransome, J., & Misra, A. (2018). *Core Software Security: Security at the Source*. CRC press.https://www.crcpress.com/Core-Software-Security-Security-at-the-Source/Ransome-Misra/p/book/9781466560956

Viega, J., & McGraw, G. (2011). "Building Secure Software: How to Avoid Security Problems the Right Way" (paperback). *Addison-Wesley Professional Computing Series*. Addison-Wesley Professional. https://www.oreilly.com/library/view/building-secure-software/9780672334092/

Unit 4: Software Vulnerabilities and Exploitation

Contents

1.0 Introduction

You will learn from this unit the detail of vulnerabilities that can be exploited by cyber-criminals to launch an attack on your computer software. You will also learn about the possible damages these vulnerabilities can cause to your computers.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- define software vulnerability
- manage the various vulnerabilities that may occur in software and how malware exploit them
- detect vulnerabilities in software using a static or dynamic method.

3.0 Main Content

3.1 What is Software Vulnerability?

Vulnerability in a computer or software is a term in cybersecurity that refers to system flaws that leave it open to attack. This vulnerability could also refer to any sort of weakness present in a computer or software, its processes or anything that makes it possible to expose information security to a threat.

Network engineers, software developers and computer users can safeguard their computers and software from vulnerabilities by updating security patches of their software regularly.

What is the use of security patches in software?

Software security patches are for solving faults or safety loopholes discovered in the original software release. Network engineers and computer users should always be kept informed of present software vulnerabilities and look for ways to safeguard against them.

3.2 Software Vulnerabilities

The most prevalent vulnerabilities of software are missing data encryption, operating system command injection, SQL injection, buffer overflow, missing authentication for critical functions, and missing authorisation. Others are unrestricted upload of dangerous file types, reliance on un-trusted inputs in a security decision, cross-site scripting and forgery, code download without integrity checks and use of broken algorithms. Similarly, URL redirection to untrusted sites, Path traversal, bugs, weak passwords and virus-infected software are common vulnerabilities. Note that, every year, the list is growing bigger as fresh ways are found by cybercriminals to steal and corrupt data.

3.3 Why Vulnerabilities in Software

Vulnerabilities of the software normally occur because programmers fail to adhere strictly to programming rules fully. Furthermore, software developers do not take elements of a computer system into considerations when designing their program, and this, in turn, creates a weakness in the system. Also, some programmers program in an unsafe and inaccurate manner that aggravates vulnerability in their software.

3.4 Software Vulnerabilities as Threat Doorways

Errors in your software can serve as a doorway to malicious threat such as malware, phishing or proxy attack in your computer or network. This is because the error has left the data in the computer or network vulnerable. With this vulnerability, cybercriminals, hackers and malware designers can take control of your computer by performing malicious activity on it such as disabling the computer, encrypting your files or data theft.

3.5 Dealing with Software Vulnerabilities

A practical scenario to dealing with vulnerabilities in software is to apply suitable security testing methods before the software is deployed. This will aid avoid vulnerabilities and attack to the software. Several techniques of software testing have been proposed and implemented in practice for dealing with software vulnerabilities. These software techniques include vulnerability coverage, source of test cases, test generation method, level of testing, and granularity of test cases, tool automation and target application.

3.6 Preventing Software Vulnerabilities

Models are the first approach to addressing and understanding vulnerabilities. However, to avoid any vulnerability related issue on your software, it is necessary to rely on standard prevention techniques for vulnerability prevention. There are two ways for the prevention of vulnerabilities on software which includes: code inspection and security activity graph.

3.6.1 Software Inspection

The method of software inspection comprises of reading or visually inspecting the program code or record of software to discover any flaws and correct them early in the phase of development otherwise it will be costly to fix the flaws once the software is deployed. A successful inspection relies on the inspector’s capacity, knowledge and type of defects he/she is trying to find.

3.6.2 Security Activity Graph

Security Activity Graphs SAGs are also useful for vulnerability avoidance. SAGs are graphical illustrations linked to causes in a VCG. SAGs show how the combination of safety operations during the design phase of software development can prevent vulnerabilities in the software.

3.7 Detecting Software Vulnerabilities

Models and inspection are helpful for understanding and preventing vulnerabilities; however, programmers also need to rely on tools to detect vulnerabilities during the software development phase. Some of these tools are based on static methods that do not need you to run a code for the detection process, while others are dynamic methods that need you to execute the code to detect vulnerability.

3.7.1 Static Methods

Static methods are those types of methods that do not require running the source code of an application. The goal is to evaluate or directly obtain specific information from the source code without executing it. There are various methods for static analysis such as pattern matching, lexical analysis, parsing, type qualifier, data flow analysis, taint analysis, model and model checking.

Pattern matching

This involves looking within the source code for a pattern string and identifying the string's number of occurrences within the source code. If for example, you consider C language, the pattern could be any call to dangerously vulnerable functions such as getc. The limitation of this method is that it produces a lot of false positives.

Lexical analysis

Lexical analysis is another static analysis method for detecting software vulnerability in which program source code is converted into a series of tokens that are subsequently compared to a vulnerability database. The limitation of this method is that it also produces a lot of false positives.

Parsing

Parsing is more complicated than lexical analysis in detecting software vulnerability; a program representation is constructed using a parsing tree to evaluate the program's syntax and semantics when the source code is parsed. This parsing method, for instance, is used to detect SQL injection attacks.

3.7.2 Dynamic Methods

In dynamic methods, it is essential to run the software program to detect vulnerabilities and then evaluate the conduct or response of the system before concluding. There are various methods for dynamic analysis such as Fault injection, fuzzy testing, dynamic taunt and sanitisation.

 **Discussion**

A company wants the software to live as soon as possible without proper check for vulnerability, how safe will you say this request is by the management.

4.0 Self-Assessment Exercise(s)

1. What is software vulnerability?
   a. Vulnerability in software refers to system flaws that leave the software open for an attack by a cybercriminal.
   b. Vulnerability in software refers to system flaws that leave the software secured to a cybercriminal.
   c. Vulnerability in software refers to system security.
   d. None among the options.

   Answer: A

Vulnerability in software refers to system flaws that leave the software open for an attack by a cybercriminal.

2. Why do vulnerabilities occur in software?
   a. Because programmers use too much security
   b. Because not all programmers are good
   c. Because programmers rarely adhere to programming rules
   d. Because programmers adhere to programming rules

   Answer: C

Vulnerabilities normally occur in software because programmers rarely adhere to programming rules, ignore computer system elements and program in an unsafe environment.

5.0 Conclusion

You have learnt from this unit that defects in software lead to vulnerabilities that can be exploited by cybercriminals to launch an attack on your computer. Network engineers, software developers and computer users should safeguard their computers and software from vulnerabilities.

6.0 Summary

In this unit, you have learnt the detail of vulnerabilities that can be exploited by cybercriminals to launch an attack on your computer software. You have also learnt about the possible damages these vulnerabilities can cause to your computers. In the next unit, you will learn about secure programming for the prevention of BOF, FSB, SQLI, XSS, session attacks.

7.0 References/Further Reading

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software Security Engineering*. Pearson India.

Axelrod, C. W. (2013). *Engineering Safe and Secure Software Systems*. Artech House.https://us.artechhouse.com/Engineering-Safe-and-Secure-Software-Systems-P1556.aspx

Fernandez, E. B. (2004, June). "A Methodology for Secure Software Design." In: *Software Engineering Research and Practice* (pp. 130-136).http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.2972&rep=rep1&type=pdf

Howard, M., LeBlanc, D., & Viega, J. (2005). *19 Deadly Sins of Software Security. Programming Flaws and How to Fix Them*.http://math.uaa.alaska.edu/~afkjm/cs470/handouts/SecuritySins.pdf

Ransome, J., & Misra, A. (2018). *Core Software Security: Security at the Source*. CRC press.https://www.crcpress.com/Core-Software-Security-Security-at-the-Source/Ransome-Misra/p/book/9781466560956

Viega, J., & McGraw, G. (2011*). Building Secure Software: How to Avoid Security Problems the Right Way* (paperback) (Addison-Wesley Professional Computing Series). Addison-Wesley Professional. https://www.oreilly.com/library/view/building-secure-software/9780672334092/

Unit 5: Secure Programming for Preventing BOF, FSB, SQLI, XSS, Session

Contents

1.0 Introduction

Today's software is implemented using different programming languages that have severe vulnerabilities that cybercriminals exploit to breach a program or a computer system. The most common vulnerabilities found in software include the buffer overflow, format string bug, SQL injection, cross-site scripting and session. These vulnerabilities are exploited in true life resulting in damages to stakeholders such as the users of the software. In this unit, you will learn the mechanism for preventing these vulnerabilities as part of software design.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- explain secure programming
- utilise BOF, FSB, SQLI, XSS, session threats to solve real-life problems
- manage the techniques for preventing BOF, FSB, SQLI, XSS, session.

3.0 Main Content

3.1 Secure Programming

Secure programming is a mechanism for creating computer software to prevent accidental vulnerability entry. Defects, bugs and logic defects are the main cause of software vulnerabilities that are frequently exploited by cybercriminals and malware designers for launching their attack on intended victims.

3.2 Software Security Vulnerabilities

Vulnerabilities in software are defects in program code resulting in safety breaches such as leakage, alteration and destruction of sensitive data. Note that cybercriminals are effective in exploring vulnerabilities. There is so much vulnerability in the software that attackers can exploit. We limit our discussion on four (4) vulnerabilities. These include the buffer overflow, format string bug, SQL injection, and cross-site scripting. These are the most frequently reported vulnerabilities in software.

3.2.1 Buffer Overflow (BOF)

Buffer overflow (BOF) enables information to be written to a program buffer exceeding the assigned size, thereby overriding the content of adjacent memory locations.

Using a buffer overflow allows an attacker to control, crash, or alter the program. There are various buffer overflow attacks categories.

Where are BOF normally encountered?

BOF is normally encountered in programs with unsafe library function calls, absence of null character at the end of buffers, pointer, and restriction to buffer access, logic mistakes and non-sufficient check before buffer access.

Types of buffer overflow attacks

Buffer overflow attacks are classified into four kinds based on buffer location in a program or process

- **Stack-based buffer overflow**: This type of buffer overflow uses a memory object known as a stack to store user input in this sort of attack. The stack is a continuous memory space used to organise function call-related data, including function parameters, local variables, and information on management, such as frame and instruction pointers.

- **Heap-based attacks:** The heap is a memory structure used to handle dynamic memory. Often programmers use the heap to allocate memory whose size is unknown at the time of compilation, where the amount of memory required is too large to fit on the stack or where the memory is intended for use across function calls. The memory space reserved for a program or process is flooded by heap attack.

- **Integer overflow attacks:** This is when an integer is used in an arithmetic operation, and the calculation outcome is a value that exceeds the integer's maximum size. Most languages of programming define maximum integer sizes. The result may cause an error if these sizes are exceeded, or it may return an incorrect result that is "wrapped around" within the length limit of the integer.

- **Unicode overflow attacks:** Unicode characters are used by attackers to generate buffer overflows in programs that expect all input to be ASCII characters.

3.3.2 Format String Bug (FSB)

Format string bug (FSB) vulnerabilities indicate the invocation of the format features with a format string that is provided by a user containing arbitrary format specifications. As a result, the number of specifiers becomes more than the number of arguments, allowing arbitrary read and write in functions and stack format.

3.2.2 SQL Injection (SQLI)

SQLI vulnerabilities can be found in programs that produce SQL queries with invalidated user inputs. The inputs may comprise arbitrary SQL queries that change the queries that were originally designed. These vulnerabilities are normally exploited by attacking SQL injection causing unexpected outcomes such as bypassing authentication and leakage of data.

3.2.3 Cross-Site Scripting (XSS)

XSS vulnerabilities enable HyperText Markup Language (HTML) content to be generated with invalidated inputs. These inputs comprise HTML tags and JavaScript code that browsers interpret as internet pages are rendered. As a consequence, the designed conduct of the generated web pages changes visibly.

3.3 Preventing Mechanism against BOF, FSB, SQLI, XSS, Session

Program security breaches are mostly blamed to programmers who overlook possible vulnerabilities in their software implementation. Moreover, the lack of understanding of a programming language implementation also contributes to writing codes that are vulnerable.

Secured programming approaches are designed to provide vulnerability free support for programming implementation and can be regarded as the first line of defence to prevent breaches of program safety. Writing safe code helps to reduce subsequent vulnerability detection and fixation expenses at later phases. Secure solutions to programming provide support in the form of secure API, libraries, aspect and filters. There are two approaches to secure programming, which includes security requirement definition and model threats.

3.3.1 Security Requirements Definitions

This involves identifying and documenting safety requirements in the early cycle of development and ensuring that subsequent development artefacts are assessed to meet those demands. If safety requirements are not defined, the resulting system security cannot be assessed efficiently.

3.3.2 Model Threats

This involves the use of threat modelling in anticipation of threats the software might face. Threat modelling includes identifying important assets, decomposing the application, identifying and categorizing threats to each asset or element, assessing threat based on a risk ranking and creating threat mitigation approaches in models, codes and test cases.

 **Discussion**

How important is security requirement definition considering the secure development process?

4.0 Self-Assessment Exercise(s)

1. What do you understand by secured programming?
   a. Secured programming is the mechanism for creating a software devoid of accidental vulnerability entry
   b. Secured programming is the mechanism for creating security codes

c. Secured programming is the mechanism for creating security protocol
d. None of the option mentioned

Answer: A

Secured programming is the mechanism for creating software devoid of accidental vulnerability entry.

2. Which among the option is a cause of vulnerability in software?
   a. Code defect
   b. Logic defect
   c. Bugs
   d. object

   Answer: D

The main causes of vulnerability in software are code defect, logic defect and bugs.

5.0 Conclusion

You have learnt from this unit that vulnerabilities found in software are defects in program code resulting in safety breaches such as leakage, alteration and destruction of sensitive data. These leakages are blamed to programmers who overlook possible vulnerabilities in their software implementation and lack of understanding of a programming language implementation.

Assignment

A software was developed for company A to take input from users, from the SDLC input validation was not considered during the development stage, List the possible attack that the software is vulnerable and explain why

6.0 Summary

At the end of this unit, you have learnt about the most common vulnerabilities found in software which include the buffer overflow, format string bug, SQL injection, cross-site scripting and session. You have also learnt the mechanism for preventing these vulnerabilities as part of software design. In the next unit, you will be learning about mobile application development security.

7.0 References/Further Reading

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software Security Engineering.* Pearson India.

Axelrod, C. W. (2013). *Engineering Safe and Secure Software Systems*. Artech House.https://us.artechhouse.com/Engineering-Safe-and-Secure-Software-Systems-P1556.aspx

Fernandez, E. B. (2004, June). "A Methodology for Secure Software Design." In: *Software Engineering Research and Practice* (pp. 130-136).http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.2972&rep=rep1&type=pdf

Howard, M., LeBlanc, D., & Viega, J. (2005). *19 Deadly Sins of Software Security. Programming Flaws and How to Fix Them.*http://math.uaa.alaska.edu/~afkjm/cs470/handouts/SecuritySins.pdf

Ransome, J., & Misra, A. (2018). *Core Software Security: Security at the Source.* CRC Press.https://www.crcpress.com/Core-Software-Security-Security-at-the-Source/Ransome-Misra/p/book/9781466560956

Viega, J., & McGraw, G. (2011). *Building Secure Software: How to Avoid Security Problems the Right Way*. (paperback)(Addison-Wesley Professional Computing Series). Addison-Wesley Professional. https://www.oreilly.com/library/view/building-secure-software/9780672334092/

Unit 6: Mobile Application Development Security

Contents

1.0 Introduction

Mobile application development security is the extent to which mobile device applications (apps) are protected from malware and cracker and other criminal operations. The word may also refer to multiple techniques and manufacturing methods that through their applications, minimize the danger of exploits on mobile devices. In this unit, you will learn the techniques for building secure mobile applications.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- manage the techniques for preventing BOF, FSB, SQLI, XSS, session
- describe the process of building a secure mobile application.

3.0 Main Content

3.1 Mobile Application Security

Mobile application security is the degree to which cell phone applications are shielded from malware, crackers and other crimes. The term may likewise allude to different advancements and practices that limit the danger of a cyber-attack on mobile devices through its application.

There are many elements in a mobile device, all susceptible to security weaknesses. Multiple players make, distribute and use these elements, each of whom plays a vital part in a device's security. Each player should integrate safety measures in the design and construction of mobile devices and the design and writing of mobile applications, but these tasks are not always performed properly. Common mobile device vulnerabilities include architectural flaws, loss or theft of devices, platform weakness, problems with isolation and permission, and application weakness.

3.2 Building Secure Mobile Application

Mobile application development is fast-growing, and with this growth pace, mobile application designers need to look at not only at offering clients with fresh characteristics but also at the application security aspect.

The security of mobile applications is one of the primary concerns these days as data residing within the application may be at risk if proper security checks are not applied during application design. Furthermore, owing to the massive use of mobile application in today’s digital globe, vulnerabilities in mobile applications have risen significantly.

Hackers are now targeting mobile applications to gain access to personal information and information about consumers to use it maliciously. Therefore, as mobile developers construct an application for bot IoS and Android platforms, they must be extra cautious.

Some of the ways to build a fully secured mobile application are discussed in the subsequent subsections.

3.2.1 Writing Secure Code

The code is the most susceptible characteristics of any mobile application that hackers can readily exploit. Therefore, writing an extremely safe code is crucial. According to studies, about 11.6 million devices are impacted by malicious code.

How do hackers make use of application code?

Hackers can reverse and engineer your application code to use it in the wrong way, so try to create a hard code that is not easy to crack and follow agile development so that you can quickly patch and update your code from time to time. In order to create the highest values of software, some of the other best is code hardening and signing.

3.3.2 Data Encryption

Encryption is the way the transmitting information can be converted into such a form that nobody else can read it without decryption. Encryption is usually done so that even if the data from the mobile application is stolen, it cannot be decrypted by the hackers, and as is the stolen data is worthless to them. Therefore, try developing an application to encrypt all the information in the application.

3.2.3 Caution Using Libraries

The mobile application developer often requires third-party libraries to build his codes. Do not trust any library to build your application as most of these libraries are not safe. When using libraries, always try testing your code to make sure your program is malware free. This is to avoid library flaws allowing malicious code to be used by cybercriminals to attack the system.

3.2.4 Authorised API Usage

Always remember to use your mobile application code with authorized APIs. It always provides the privilege of using your data to hackers. For instance, hackers can use the authorisation data caches to obtain system authentication.

3.2.5 Use Good Authentication

Authentication mechanisms are the most important components of safety for mobile applications. One of the top vulnerabilities in mobile applications is weak authentication. As a developer and user, from a safety point of view, authentication should be regarded essential. Password is one of the most common methods of authentication, so password should be strong enough so that it cannot be broken.

3.2.6 Develop Temper Detection Techniques

This technique is used to receive an alert when changing or modifying your code. Often, a log of code modifications to your mobile application is vital to prevent a malicious programmer from injecting poor code into your application. Try to have triggers intended to maintain activity logs for your application.

3.2.7 Least Privileges

The least privilege principle is often essential for the safety of the mobile application code. Only those who intend to receive them should have access to the code, and the rest should not be given the privileges, keeping it to a minimum.

3.2.8 Session Management

Session handling is significant in mobile application development as mobile sessions are generally longer than desktop sessions. Session management should, therefore, be performed to preserve the safety of stolen and lost devices and should be performed using tokens rather than identifiers.

3.2.9 Repeated Testing

Repeated testing for modification is the easiest solution for a secure mobile development application. This is because safety elements change every day. To safeguard, your mobile application, you need to be updated with safety trends. You should opt for penetration testing and emulators to get an idea of your mobile application vulnerabilities so they can be eliminated or reduced.

Assignment

Encryption is important for the transmission of sensitive data, Explain the importance of encryption and possible attack against that can affect the process of data transmission.

4.0 Self-Assessment Exercise(s)

1. What are the two ways of creating a high-value software code in a mobile application?
   a. Code hardening and softening
   b. Code hardening and signing
   c. Code Softening and signing
   d. None of the mentioned option
   Answer: B

   The two best ways to create the highest values of software are to use code hardening and signing.

2. Encryption is usually done so that even if the encrypted data is stolen, it cannot be decrypted by the attackers, which makes the stolen data worthless to them.
   a. True
   b. False
   Answer: A

5.0 Conclusion

You have learnt from this unit that it is important to build secure mobile applications whose component are safely guarded against security vulnerabilities and cyber-criminal attacks such as malware, cracker and other criminal operations.

6.0 Summary

At the end of this unit, you have learnt the techniques for building a secured mobile application devoid of vulnerabilities.

7.0 References/Further Reading

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software Security Engineering.* Pearson India.

Ammann, P., & Offutt, J. (2016). *Introduction to Software Testing*. Cambridge University Press.https://cs.gmu.edu/~offutt/softwaretest/

Axelrod, C. W. (2013). *Engineering Safe and Secure Software Systems*. Artech House.Https://Us.Artechhouse.Com/Engineering-Safe-And-Secure-Software-Systems-P1556.Aspx

Fernandez, E. B. (2004, June). "A Methodology for Secure Software Design." In: *Software Engineering Research and Practice* (pp. 130-136). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.2972&rep=rep1&type=pdf

Howard, M., LeBlanc, D., & Viega, J. (2005). *19 Deadly Sins of Software Security. Programming Flaws and How to Fix Them.* http://math.uaa.alaska.edu/~afkjm/cs470/handouts/SecuritySins.pdf

McGraw, G. (2006). *Software Security: Building Security In*. (Vol. 1). Addison-Wesley Professional.https://www.oreilly.com/library/view/software-security-building/0321356705/

Ransome, J., & Misra, A. (2018). *Core Software Security: Security at the Source*. CRC press.https://www.crcpress.com/Core-Software-Security-Security-at-the-Source/Ransome-Misra/p/book/9781466560956

Viega, J., & McGraw, G. (2011). *Building Secure Software: How to Avoid Security Problems the Right Way.* (paperback)(Addison-Wesley Professional Computing Series). Addison-Wesley Professional. https://www.oreilly.com/library/view/building-secure-software/9780672334092/

Module 3: Design and Testing for Security, Best Practices

Module 3: Design and Testing for Security, Best Practices

Introduction

In software security, the threat is often anybody who intends to harm using some software agents. In most cases, the software can be subject to either threat during development or threat during operation. Also, weaknesses likely to be targeted are those found in the software components' external interfaces, because those interfaces provide the attacker with a direct communication path to the software's vulnerabilities. Several well-known attacks target software that incorporates interfaces, protocols, design features, or develops faults that are well understood and widely publicised as harbouring inherent weaknesses. It is therefore very important to consider security right from early design through to the implementation and testing phases of the software development life cycle.

In this module, you are going to learn how to effectively perform continuous auditing and monitoring to maintain the security state of your organization’s information system. You are going to learn several architectural and design principles for developing a secure software system, as well as several techniques and methods for analysing and identifying architectural flaws, and development bugs. You are also going to learn several source code analysis techniques, including static code review and security testing.

The module is organised into three units as follows:

Unit 1: Secure Software Design Principles
Unit 2: Static Analysis Techniques
Unit 3: Security Testing

Unit 1: Secure Software Design Principles

Contents

1.0 Introduction

In this unit, you are going to learn different practices for designing secure software; you are also going to learn fundamental design principles that you can adopt in designing secure software application.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- describe the principles of secure software design
- explain the concept of secure characterisation
- apply the Software security design principles to maintain confidentiality, integrity, and availability of a system, sub-system, and system data.

3.0 Main Content

3.1 Overview of Secure Software Design Principles

Software design phase translates and transforms ideas into reality. The *what* and *why* of the requirements phase becomes *who*, *when*, *where* and *how* of the software. This phase is as critical as the requirements phase in terms of contribution towards overall success and quality of the eventual

deliverable, from a functionality perspective. In terms of security, architecture and design are seen by many as the most critical phase of the Software Development Life Cycle (SDLC). This is because, good design decision yields an approach and structure that are resistant and resilient to attack, and also, prescribe and guide good decision in later phases, for instance, coding and testing. However, bad design leads to flaws that may be difficult to overcome by even most disciplined code and test efforts.

Although much security efforts are put in tackling implementation challenges, such as buffer overflow, sql injection, and other bugs, the reality, however, is that most defects that lead to security vulnerabilities in software are attributed to architectural and design flaws. These flaws tend to have a greater footprint in terms of potential security impact and exploit within a piece of software and potentially across multiple systems and projects. Hence, the goal of integrating security into the design phase of SDLC is to reduce the number of flaws significantly, as early as possible while also minimising ambiguities and other weaknesses.

3.2 Secure Software Practices for Design

To integrate security into the design phase of SDLC, the practice of risk analysis is important. Architectural risk analysis ensures that security concerns of architectural and design-level are identified and addressed early in the SDLC. This results in an improved attack resistance level, tolerance and resilience.

Risk analysis identifies potential threats, and the identified threats are mapped to the risks they bring. These risks are the likelihood of a given threat exploiting a particular vulnerability, with the impact the harmful event may have on the asset. Threats to the software must be analysed to understand the likelihood of an attack; you also analyse potential vulnerabilities, as well as security controls in place for the software. The risk analysis methodology is discussed below:

- **Software Characterisation**

The first step of any software risk analysis is to understand what the kind of software, and how it works. For design risk analysis, the minimal understanding required is the system's description using high-level diagramming techniques. A common diagramming format that has proven itself to be effective for architectural risk analysis is a whiteboard-type, high-level, one-page diagram that shows how components are connected together, as well as how control and data flow are managed. The diagram is important in identifying architecture and design-level flaws that are hard to detect during the code-level review.

To gather information for software characterisation, a wide spectrum of system's artefacts are reviewed, interviews are also conducted with key stakeholders such as program/product managers, and software architects. Some of the artefacts to review for software characterization includes:

- Business case of the Software
- Functional and nonfunctional requirements
- Requirements enterprise architecture
- Use case description documents
- Misuse or abuse case documents
- Software architectural documents that describe physical, logical, and process views
- Data structure documents
- Design documents including UML diagrams showing structural, and behavioural aspects of the system
- Risk management plan
- Software acceptance plan
- Problem resolution plan
- Transactions security architecture documents
- Configuration and change management plan
- Identity services and management architecture documents

The goal of software characterisation activity is to produce documents depicting vital relationships among critical parts of the system. Figure 3-1 shows an example of a high-level software architecture diagram. The diagram shows major system components, interactions among components, and various zones of trust (areas that share common level and management mechanism of privilege). Avatars and their associated arrows show potential attackers with attack vectors against the system. The potential threats and attack vectors are further broken down in more details during the subsequent stages of architectural risk analysis.

- **Threat Analysis**

Threats violate the protection of information assets and security policy. Threat analysis aims at identifying relevant threats for a particular functionality, architecture, and configuration. During this stage, vulnerabilities are mapped to the identified threats, understand how the software may be exploited. The mitigation plan is developed, that consists of countermeasures considered effective against identified vulnerabilities.

Why is architectural vulnerability assessment important in secure application development?

- **Architectural Vulnerability Assessment**

Vulnerability assessment examines the conditions that must exist before vulnerabilities can be exploited, and assesses the state of the software after exploitation. There are three (3) activities for architectural vulnerability assessment, as discussed below:

1. **Attack Resistance Analysis**: This is a process of inspecting the software design to find common weaknesses, which could lead to vulnerabilities and susceptibility of the system to common attack patterns. To perform attack resistance analysis, you consider your architecture and compare it with some known bad practices, as well as known good practices. For instance, you may consider the *principle of least privilege* (discussed in section 3.3), which states that all operations on the software should be done with the least possible privilege required for that operation. Here, you find areas in the software that operates at an elevated level of privilege. You can achieve this by perhaps drawing the system's major components, classes, and subsystems. You then circle the areas of high privilege against areas of low privilege. You then consider the boundaries between identified areas and understand the type of communications that happens across the boundaries. Once these vulnerabilities are identified, you map the relevant attack patterns against the architecture, with special consideration given to areas of identified vulnerability. Finally, you should capture and quantify any attack that found to be viable against identified vulnerabilities as a risk to the software.

2. **Ambiguity Analysis: Ambiguity** is a source of vulnerability if it exists between specifications and development. Design plays an important role in eliminating potential misunderstandings between business requirements and the implementation of the software's actions. During this activity, you examine all artefacts that define software's functions, structure, properties and policies for any ambiguity in the description that may lead to multiple interpretations. This is because multiple interpretations constitute a risk to the software. At this stage, you take note of places where either the requirements or architecture are defined ambiguously. You may also note places where architecture and implementation disagree or fail to resolve the ambiguity. Consider a scenario where a "requirement for web application states that an administrator can lock an account such that user can no longer log in while the account remains locked". Now, this requirement raises some important questions; for instance, what will happen to sessions of a user who is an active administrator locks the account? Is the user forcibly logged out? Or does active session remain valid until the user logs out? To answer these questions, the authentication and authorisation architecture must be compared with the actual implementation in an existing system. You should balance the security consequences that continue to exist even after the account has been locked against sensitive information asset you are trying to protect.

3. **Dependency Analysis:** This activity involves analyzing vulnerabilities associated with the execution environment of the software. It may include vulnerabilities associated with the operating system, network vulnerabilities, and interaction vulnerabilities that emerge from components interaction. Dependency analysis is aimed at developing a list of software or system vulnerabilities that could be triggered accidentally or intentionally, that leads to a violation of security policy or security breach.

What would you use as metrics to estimate the likelihood of a risk?

- **Risk Likelihood Determination**

This is a qualitative estimate likelihood of a successful attack. It is done based on analysis, as well as past experience. Three factors described below can be used to estimate the likelihood of risk.

- Threat's motivation and capability, e.g. a paid hacker who is paid by the criminal organisation to hack is more highly motivated than a college student that hack for fun.
- Vulnerability impact
- Effectiveness of current control, e.g. compromising two-factor authentication systems can be harder than compromising simple user IDs and passwords.

You can estimate risk likelihood by rating it as ***High***, when all three factors are weak (threat is capable and motivated, there is severe vulnerability, and prevention controls are not very effective). ***Medium***, when one of the three factors is compensating, but the other two are not. ***Low***, when at least two of the three factors are compensating.

- **Risk Impact Determination**

This activity analyses the consequences a business may face if the worst-case scenario happens. Three aspects of risk impact determination are discussed below:

- ***Identify Threatened Assets:*** You should identify assets threatened by a realisation of the risk; you should also identify the nature of what will happen to these assets.
- ***Identify Business Impact:*** Here, you identify the extent to which business will suffer if an attack takes place.

- **Risk Mitigation Planning**

Mitigation of risk involves changing the software architecture in one way or the other to reduce the risk likelihood or its impact. Several measures can be taken to reduce the likelihood of risk, as discussed below:

- Raising the bar in terms of skill required to exploit the vulnerability. For instance, you may change the authentication mechanism from

IDs and passwords to pre-shared public key certificates. This could make impersonation very difficult.

- Reduce the period from which vulnerability is available for exploit. For instance, if the session expires after 15 minutes of inactivity, then the opportunity window for session hijacking is about 15 minutes.
- Cryptography could help when correctly applied. For instance, detecting corruption in encrypted data is easier than in unencrypted data, and encrypted data is more difficult for an attacker to use.

You are to develop a software application for a bank to process an important document, what are the secure design principle to consider.

3.3 Basic Secure Design Principles

Secure Design Principles are fundamental truth upon which software is built in order to be robust against attack. These high-level practices are derived from real-world experience to guide software developers in building secure software. Leveraging these principles can benefit the development team with the industry's leading practitioners guidance and learn to ask the right questions of their software architecture and design in order to avoid the most serious flaws. The basic principles are discussed below and depicted in figure 6.

- ***Principle of Least Privilege:*** This principle states that each component in the system should be allocated minimum necessary right to fight its functions, nothing more. This principle is effective in reducing insecurity influence of component failure and effective in reducing the effort required in security evaluation of the component.

Fig. 6: Basic Secure Design Principles

- ***Principle of Fail Safe and Fail Secure:*** This principle states that when the system fails, the system's mechanism or function should not lead violation of security policy. Preferably, the system should detect a failure at any point during operation, including

initialization, maintenance, normal operation, shutdown, error detection, and recovery.

- ***Least common mechanism:*** The principle states that mechanisms of the system should not be shared by the user except when necessary. Since shared mechanisms sometimes lead to unintended means of interference.
- ***Separation of Duties:*** The principle requires the user to have multiple privileges to accomplish a large range of security compromise. Programs and systems should grant access to assets only when the entire condition is met. It provides control over the resources and extra guarantee that the system access is authorised.
- ***Simplicity:*** The principle states that complexity does not add security. The simpler the system, then less can go wrong because when an error happens, it becomes simpler to understand and fix.
- ***Secure Weakest Link:*** Attackers often targets the weakest link in the system. To ensure secure design, identify the weakest point of compromise in the system. The weakest point is, mostly, trusted third-party components. Ensure these points are secure enough.
- ***Defence in Depth:*** This principle describes the defence derived from implementing multiple defence mechanisms.
- ***Trust Nobody:*** This principle states that all employees, users, and other third parties, may turn malicious to the system. Hence, don't trust them completely.
- ***Leverage Existing Components:*** The principle promotes the concept of reuse of existing secure components. It states that using existing, well-tested resources safer than developing new components.

4.0 Self-Assessment Exercise(s)

1. One of the secure design principles suggests that Granting permissions to a user beyond the scope of the necessary rights of action can allow that user to obtain or change the information in unwanted ways. Which principle is it?
   a. Least common mechanism
   b. Separation of duties
   c. Principle of least privilege
   Answer: B

2. Which principle suggests that defending an application with multiple layers can eliminate the existence of a single point of failure that compromises the security of the application?
   a. Simplicity
   b. Defence In Depth
   c. Leverage Existing Components
   Answer: B

5.0 Conclusion

The architecture and design phase of the SDLC is a critical stage for identifying and preventing security flaws before they become part of the software. As the connectivity, complexity, and extensibility of software increase, the importance of addressing security concerns as an integral part of the architecture and design process becomes even more critical. During this phase, designers and security analysts leverage security design practices to ensure that requirements are appropriately interpreted to give the software structure and form in a way that minimises the security risk.

Assignment

A company wants to know what to consider to build a secure web application, list, and explain the steps that you will recommend base on what you have learned in this unit.

6.0 Summary

In this unit, you should get the following key points:

- Good design decision yields an approach and structure that are resistant and resilient to attack.
- Software characterisation, threat analysis, architectural vulnerability assessment, risk likelihood determination, risk impact determination, and risk mitigation planning are effective design risk analysis methodologies for identifying potential design flaws and identifying their control mechanisms.
- Secure design principles are high-level practices that are derived from real-world experience to guide software developers in building secure software.
- Leveraging these principles can benefit the development team with the industry’s leading practitioners' guidance and learn to ask the right questions of their software architecture and design to avoid the most serious flaws.

7.0 References/Further Reading

McGraw, G. (2006). *Software Security: Building Security In.* Boston, MA: Addison-Wesley, https://www.oreilly.com/library/view/software-security-building/0321356705/

Merkow, M., & Raghavan, L. (2010). *Secure and Resilient Software Development*. New York: Taylor & Francis Group.https://doi.org/10.1201/EBK1439826966

Unit 2: Static Analysis Techniques

Contents

1.0 Introduction

In this unit, you are going to learn about different implementation level defects. You will also learn how to discover these defects, following manual source code review process. Finally, you will know more about how to do static code analysis, with different automated static analysis tools.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- explain the overview of code analysis
- analyse and debug software code without executing it
- identify common software vulnerability.

3.0 Main Content

3.1 Overview of Code Analysis

Most attacks on software applications arise because of defects in its design, coding, testing, as well as operation. Defects often fall under any one of the following two categories:

- A Bug, which is a problem introduced during implementation. Examples of bug include race condition, buffer overflow, unsafe system calls, and incorrect input validation.

- A Flaw, which is a problem at a much deeper level. They were mostly originating from design and instantiated in the code. Examples are compartmentalisation problems in the design, error-handling problems, and broken or illogical access control.

Software security problems are divided 50/50 between bugs and flaws. Therefore, discovering and eliminating bugs during code analysis takes care of roughly half the problem of software security. Code analysis focuses on addressing implementation level bugs.

3.2 Common Software Code Vulnerabilities

Some of the common security implementation bugs are discussed below:

- Incorrect or incomplete input validation: Inadequate input validation leads to several bugs, such as buffer overflow and SQL injection, which can compromise the integrity and confidentiality of the system.
- Exception handling: Exceptions are events that disrupt the normal flow of code. Security defects related to error handling are very common. A bug could emerge when you forget to handle errors or were handled roughly - for instance, having unchecked error value and empty catch block.
- Buffer overflow: Buffer overflows are very common bugs used to exploit software by injecting malicious code remotely into a target software.
- SQL injection: Developers often chain SQL commands together with user-provided parameters. Hence, attackers exploit this and embed SQL commands to these parameters and execute arbitrary SQL queries or commands on the database server through the application.
- Race conditions: Race conditions are often characterised as scheduling dependencies between multiple threads that are not synchronised properly, which may cause undesirable timing of events. Security concern may arise when specific events sequence is needed between, say Event X and Event Y, but a race occurs, and the proper sequence is not ensured by the program. Example of race condition bug includes deadlock, infinite loop, and resource collision.

3.3 Source Code Review

Source code review for security is an effective practice that enhances software security. Code review is about finding and fixing the bug. It involves reviewers meeting one-on-one with developers to visually review code and check whether it meets secure code development criteria. The

code review process consists of four high-level steps, as shown in figure 7.

First step of the review process involves understanding the purpose of the application, internal design of the system, and the threat models prepared for the application. This step is important for identifying critical components of the code and assigning priorities to them. Prioritising is important, as full coverage may be unrealistic. Hence most critical components should be reviewed completely.

Second step involves reviewing the identified critical components according to their priority. All the identified defects should be documented, and appropriate priorities should be assigned. Reviewers then should document these defects together with suggested fix approaches for each to ensure they do not creep into the final production code.

Third step of a code review is implementing the fixes for the problems revealed in the review. It may involve integrating an existing, reusable security component available to developers, or may require simple or even complex changes to the code and subsequent reviews.

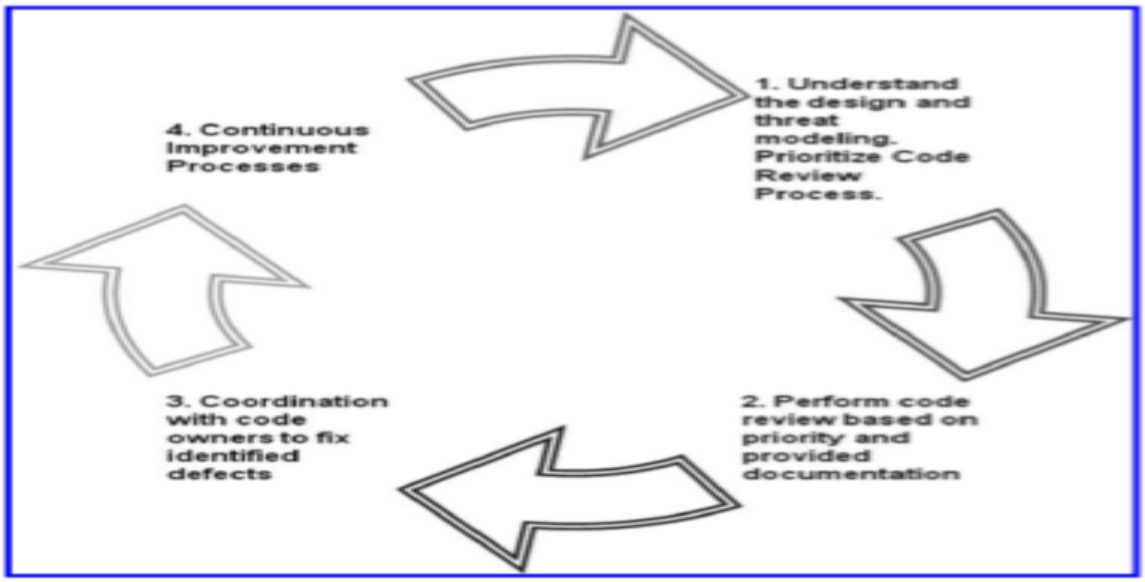

Fig. 7:The Code Review Process
(Source: Slideshare.net)

The final step involves studying the lessons learned during the review cycle and identifying areas for possible improvements. This ensures that the next code review cycle is more efficient and effective.

Examples of critical components that require an intense review and analysis are user authentication and authorisation, data protection

routines, code involved in handling error conditions, data validation routines, code that receives and handles data from untrusted sources, usage of operating system resources and networks, usage of problematic/deprecated APIs, etc.

What is the difference between manual code review and automated static code analysis?

In a manual code review, reviewers and developers visually inspect the code to determine whether it meets previously established secure code development criteria. In automated static code analysis, developers automate the analysis of source code by using static analysis tools.

Manual code inspection for security vulnerabilities is time-consuming. For it to be effective, reviewers must know what security vulnerabilities look like before they can examine the code rigorously and identify those problems. For more effective and faster analysis, automated static analysis tools can be used to complement manual reviews. These tools can evaluate software programs more frequently; they can also encapsulate security knowledge in a way that operator of the tool does not need to have the same level of security expertise as a human reviewer. Example of static code analysis tools are:

- Armorise CodeSecure (A commercial, multi-language with built-in language parsers for analyzing C#, java, .NET, PHP, etc.).
- Coverity software integrity, which identifies vulnerabilities and defects in C, C++, C#, and Java.
- RATS (Rough Auditing Tool for Security), An open-source multi-language analyzer.
- Checkstyle, scans java code, also shows a violation of configured coding standard.

3.3.1 Automated Static Code Analysis

Static source code analysis is the process by which developers check their code for problems and inconsistencies before it is compiled. Automating it involves the use of tools to scan the source code and automatically detecting errors that typically pass through compilers and can cause problems later in the SDLC. These tools generate reports that present analysis results graphically, and recommend potential resolutions to the identified problems.

Major advantage of static analysis is that it is performed before a program reaches completion level, where dynamic analysis could be used. However, static analysis too should not be considered as a panacea to all potential problems. You should keep in mind that they can produce false positives and false negative. Therefore, you should not consider results indicating zero security defects to mean that your code is completely free

of vulnerabilities or 100 per cent secure. Instead, these results simply mean that your code has none of the patterns found in rule-base of the analysis tool for security defects. These tools look for a fixed set of patterns or rules in the code in a manner similar to virus-checking programs. While some of the more advanced tools allow new rules to be added to the rule-base, the tool will never find a problem if a rule has not been written for it. Examples of problems static code analysers can detect are as follows:

- Syntax problems
- Unconditional branches into loops
- Unreachable code
- Parameter type mismatches
- Undeclared variables
- Non-usage of function results
- Uninitialised variables
- Uncalled functions and procedures

Static code analysis can be used to discover subtle and elusive implementation bugs before the software is tested or placed into operation. Correcting these errors earlier in the code could reduce testing efforts, and costs of operations and maintenance will be minimised.

4.0 Self-Assessment Exercise(s)

1. Which of the following is NOT an example of an implementation bug?
   a. compartmentalisation problems in the design
   b. unsafe system calls
   c. illogical access control

   Answer: A

2. Which of the following problems can static code analysers detect?
   a. Undeclared Variables
   b. Buffer Overflow
   c. Architecture flaws

   Answer: C

5.0 Conclusion

We discussed that software defects cause the majority of software vulnerabilities, and understanding the sources of vulnerabilities and learning to program securely is essential for protecting the software from

attack. Common software bugs discussed in this unit include incomplete or incorrect input validation, improper exception handling, sql injection, buffer overflow, etc. Discovering and eliminating these bugs takes care of roughly half the problem of software security. Code analysis detects implementation level bugs. Several tools are available that automates static code analysis, by scanning the code and generate reports that present analysis result, with recommendations for potential resolutions to the identified problems. We have seen that source code review could reduce testing efforts, and minimise costs of operations and maintenance.

6.0 Summary

In this unit, you should get the following key points:

- Software security problems are divided between bugs and flaws.
- Discovering and eliminating bugs solves roughly half the problem of software security.
- Code analysis focuses on addressing implementation level bugs.
- The first step in the code review process involves understanding the application's purpose, its internal design, and the threat models prepared for the application and followed by reviewing the identified critical components according to their priority. The third step implements the fixes for the problems revealed in the review, followed by studying the lessons learned during the review cycle, and identifying areas for possible improvements.
- Static code analysis involves the use of tools to scan the source code and automatically detecting errors that typically pass through compilers and can cause problems later in the SDLC.

7.0 References/Further Reading

Allen, J., Barnum, S., Ellison, R., McGraw, G., & Mead, N. (2008). *Software Security Engineering*. Massachusetts: Addison Wesley Professional.

Allen, J. H.; Barnum, S. J.; Ellison, R. J.; McGraw, G. & Mead, N. R. (n.d.). *Software Security Engineering: A Guide for Project Managers.* Pearson Education ISBN-10:032150917X•ISBN-13:9780321509178.

McGraw, G. (2006). *Software Security: Building Security In.* Boston, MA: Addison-Wesley, 2006.https://www.oreilly.com/library/view/software-security-building/0321356705/

Unit 3: Security Testing

Contents

1.0 Introduction

In this unit, you will learn penetration testing using white box approach, and black box approach. You will be introduced to some tools used for black box testing.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- discuss security testing on software using the black box and white box
- explain the difference between software testing and software security testing.

3.0 Main Content

3.1 Overview of Security Testing

Security testing is mainly performed to show that software meets its security requirements, identify and minimise the number of security vulnerabilities in the software before going into production. Security testing reduces the overall cost of the project, protects the reputation of

the organisation, reduces litigation expenses, and it helps in making sure that the product complies with regulatory requirements.

The overall goal of security testing is ensuring the software is robust and continues functioning acceptably even under malicious attack. Security testing is influenced through probing undocumented assumptions and regions of specific complexity to determine how a software program can be broken. Designers and the requirement engineers would possibly outline a secure design, and developers would probably be diligent and develop secure code. Still, in the end, the testing method determines whether the software is adequately secured as soon as it is fielded.

3.1.1 Software Testing vs Software Security Testing

A common misconception about bugs was that security bugs in a software system were similar to traditional programming bugs and traditional testing and quality assurance techniques could be used for secure software development. However, developers have learned over time, that security-related bugs could differ from traditional software bugs in many ways. These traits, in return, influence the practices that you apply for software security testing.

Security testing is different from conventional software testing because its focus is on what the software *should not* do rather than what it *should* do. Although in a few occasions, it also tests conformance to positive requirements such as "User accounts are disabled after three unsuccessful login attempts" and "Network traffic must be encrypted”. In most cases, however, it tests negative requirements such as "Outside attackers should not be able to modify the contents of the Web page" and "Unauthorised users should not be able to access data." Its focus on negative requirements rather than positive affects the way this testing is conducted. The best way to perform positive requirement testing is to have a test case where the requirement is intended to be true and verify the requirement is satisfied by the software. In contrast, the negative requirement could state that something ought to have never occurred. To perform standard testing approach to negative requirements, one would need to create every possible set of conditions, which is not feasible.

Common software security testing method is penetration testing that involves attacking the software to analyse its behaviour as a result of that attack. Penetration testing and its approaches are explained in section 3.2.

Who is responsible for performing penetration testing?

3.2 Software Penetration Testing

Security testing is often called ***Penetration Testing (PenTest)***. It is aimed at demonstrating that software meets specified security requirements, and to identify and minimise the number of security vulnerabilities in the software before it goes into use. Penetration testing is performed outside the attacker's perspective (someone without inside knowledge of the application) and involves exploiting identified vulnerabilities to break the system or gain unauthorised access to information. Its aim is not only limited to identifying potential vulnerabilities but also to determine the exploitability of an attack and the degree of impact of a successful exploit.

There are several activities involved in performing penetration testing, which varies according to the application type. Figure 8 shows a sequence of fundamental penetration testing activities that are common to most applications.

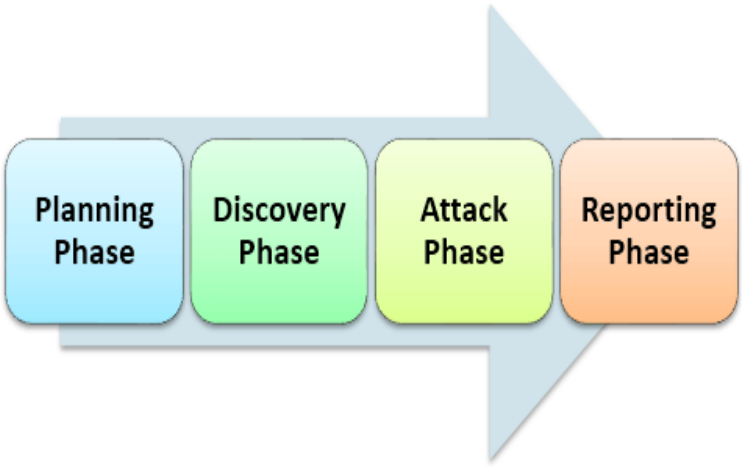

Fig. 8: Fundamental Penetration Testing Activities

The first activity is the **planning phase,** which involves defining the scope of the test, and defining the strategy that will be used for the test. To define the scope of the test, you use the existing security policies as well as standards of the organisation. The second activity is the **discovery phase** that involves fingerprinting, that is collecting information about the system, which includes the system's data, usernames and passwords. It also involves checking for system vulnerabilities. **Attack phase** involves finding exploits for all identified or discovered vulnerabilities. The final activity is **reporting phase,** which involves a detailed report of all findings, risks of the found vulnerabilities with their impact, as well as recommendations and solutions.

Common penetration testing approaches are white box and black box approaches discussed in sections 3.2.1 and 3.2.2.

3.2.1 Black box Penetration Testing

Black box testing involves a set of activities that occurs during the pre-deployment test phase or periodically after a system has been deployed. In this approach, a running program is analyzed and probed with various inputs. It requires running a program only and doesn't use source code analysis of any kind. Malicious input is supplied to the program to test an application with this approach. The intention is to break it. If the program breaks during a particular test, then the security problem might have been discovered.

Black box penetration testing is automated, and some of the few commercially available black box penetration testing tools and suites are:

- Cenzic Hailstorm (http://www.cenzic.com)
- HPWebInspect (https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-201200^9570_4000_100__)
- IBM AppScan (http://www-01.ibm.com/software/awdtools/appscan)

Typically, these tools look for and report on the following vulnerabilities:

- Improper input validation
- Command injection and buffer overflow attacks
- SQL injection attacks
- Cross-site scripting vulnerabilities
- Cross-site request forgeries
- Directory traversal attacks
- Improper session management
- Improper authorisation and access control mechanisms

3.2.2 White Box Penetration Testing

White box analysis involves analysing and understanding both the source code and the design. White box testing is very effective in finding programming errors, and in some cases, it amounts to pattern matching, often automated with a static analyser (discussed in unit 2). The major drawback with this approach is that it can produce a false positive or false negative result, as discussed in unit 2. Some of the tools available for white box testing and bugs that they detect were discussed in Unit 2.

Regardless of whether a black or white box approach is used, any testing method can reveal possible software risks and potential exploits. One problem with almost all kinds of security testing is the lack of it. Most organisations focus on features and spend very little time understanding or probing nonfunctional security risks. Additionally, penetration testing may not find all vulnerabilities in the system, and there may be some limitations ranging from time, budget, scope, as well as skills of the testers.

Discussion

A black box penetration testing is very good and fast, in what situation will you consider a white box white box over black box penetration testing.

4.0 Self-Assessment Exercise(s)

1. Which of the following is Not true about penetration testing?
   a. It is being done on the entire system.
   b. It is aimed at proving that defects do not exist.
   c. It is a deliberate attack on the application.

   Answer: B

2. Which of the following vulnerabilities can be looked at by black box penetration testing tools?
   a. Cross-site scripting vulnerabilities
   b. Directory traversal attacks
   c. All of the above

   Answer: C

5.0 Conclusion

Certain activities relevant to software security, such as stress testing, are often carried out at the system level. Penetration testing is also carried out at the system level, and when a vulnerability is found in this way, it provides tangible proof that the vulnerability is real. A vulnerability that can be exploited during system testing will be exploitable by attackers. In the face of schedule, budget, and staff constraints, these problems are the most important vulnerabilities to fix. Black box approach requires running a program, without knowledge of the source code, while providing malicious inputs to break it. These testing methods can reveal possible software risks and potential exploits.

Mini Project.
As a software security engineer, base bank of Nigeria approached you with their planned mobile banking application they are about making public for customers to use. They want you to come up with an application testing report considering the following test cases and possible attack at each stage of the test case. Tools to be used in each case should listed.

- ✓ As a software security engineer, base bank of Nigeria approached you with their planned mobile banking application they are about making public for customers to use. They want you to come up with an application testing report considering the following test cases and possible attack at each stage of the test case. Tools to be used in each case should listed. (20 Marks)
- ✓ Authentication & Authorization
- ✓ Information Disclosure
- ✓ Session Management
- ✓ Input Validation
- ✓ Error Handling
- ✓ Cryptography
- ✓ Application Process Flow Review
- ✓ Application Technical Vulnerability Exposures.

Note:
Your recommendation should be the solution to the above listed issues.

6.0 Summary

In this Unit, you should get the following key points:

- Security Testing demonstrates that software meets specified security requirements.
- White box testing is effective in detecting programming errors, and in some cases, amounts to pattern matching, often automated with a static analyser.
- Both black and white box approaches can reveal possible software risks and potential exploits.

7.0 References/Further Reading

Allen, J. H.; Barnum, S. J.; Ellison, R. J.; McGraw, G. & Mead, N. R. (n.d.). *Software Security Engineering: A Guide for Project Managers.* Pearson Education ISBN-10:032150917X•ISBN-13:9780321509178.

Allen, J., Barnum, S., Ellison, R., McGraw, G., & Mead, N. (2008). *Software Security Engineering*. Massachusetts: Addison Wesley Professional.

McGraw, G. (2006). *Software Security: Building Security In*. Boston, MA: Addison-Wesley, 2006.https://www.oreilly.com/library/view/software-security-building/0321356705/

Merkow, M., & Raghavan, L. (2010). *Secure and Resilient Software Development.* New York: Taylor & Francis Group.https://doi.org/10.1201/EBK1439826966.