SAD LAB EXPERIMENT - 2

Aim: To learn Case Study for SDLC

To do:

1. What is a secure SDLC? Why is it important?
2. Phases involved in secure SDLC.
3. Case Study on SDLC.

Theory:

SDLC

What is Secure SDLC?

The Software Development Lifecycle (SDLC) is a structured process which enables high-quality software development, at a low cost, in the shortest possible time. Secure SDLC (SSDLC) integrates security into the process, resulting in the security requirements being gathered alongside functional requirements, risk analysis being undertaken during the design phase, and security testing happening in parallel with development, for example.

Secure SDLC processes dovetails with DevSecOps, and works in all delivery models from the traditional waterfall and iterative, to the increased speed and frequency of agile and CI/CD.

Secure Software Development Lifecycle brings security and testing into each development stage:

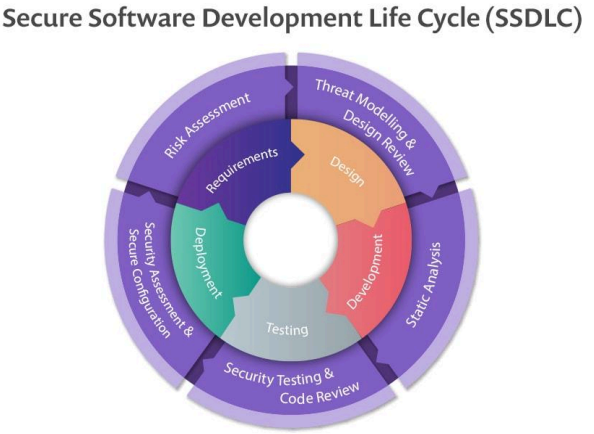
- **Planning:** This stage in the Secure SDLC means collating security inputs from stakeholders alongside the usual functional and non-functional requirements, ensuring security definitions are detailed and embedded from the outset.
- **Development:** Product development is enhanced by Secure SDLC with security best practices leveraged to create code that is secure by design, as well as establishing static code review and testing in parallel with development to ensure this is the case.
- **Build:** Secure SDLC demands that the processes used to compile software also be monitored, and security assured.
- **Testing:** Testing throughout the lifecycle is critical to Secure SDLC, and now includes assurance that all security requirements have been met as defined. Test automation and continuous integration tooling are essential to a functional Secure SDLC.
- **Release and Deploy:** The release and deploy lifecycle stages are bolstered by Secure SDLC, with additional monitoring and scanning tooling deployed to ensure software product integrity is maintained between environments. CI/CD pipelines automate secure and consistent delivery.

Why is Secure SDLC Important?

Secure Software Development Lifecycle seeks to make security everybody's responsibility, enabling software development that is secure from its inception. Put simply, Secure SDLC is important because software security and integrity are important. It reduces the risk of security vulnerabilities in your software products in production, as well as minimising their impact should they be found.

Gone are the days of releasing software into production and fixing bugs as they are reported. Secure Software Development Lifecycle puts security front and centre, which is all the more important with publicly available source code repositories, cloud workloads, containerization, and multi-supplier management chains. Secure SDLC provides a standard framework to define responsibilities, increasing visibility and improving the quality of planning and tracking and reducing risk.

Phases of Secure Software Development Life Cycle



Each phase of the SDLC must contribute to the security of the overall application. This is done in different ways for each phase of the SDLC, with one critical note: Software development life cycle security needs to be at the forefront of the entire team's minds. Let's look at an example of a secure software development life cycle for a team creating a membership renewal portal:

Phase 1: Requirements

In this early phase, requirements for new features are collected from various stakeholders. It's important to identify any security considerations for functional requirements being gathered for the new release.

- **Sample functional requirement:** user needs the ability to verify their contact information before they are able to renew their membership.
- **Sample security consideration:** users should be able to see only their own contact information and no one else's.

Phase 2: Design

This phase translates in-scope requirements into a plan of what this should look like in the actual application. Here, functional requirements typically describe what should happen, while security requirements usually focus on what shouldn't.

- **Sample functional design:** page should retrieve the user's name, email, phone, and address from CUSTOMER_INFO table in the database and display it on screen.
- **Sample security concern:** we must verify that the user has a valid session token before retrieving information from the database. If absent, the user should be redirected to the login page.

Phase 3: Development

When it's time to actually implement the design and make it a reality, concerns usually shift to making sure the code is well-written from the security perspective. There are usually established secure coding guidelines as well as code reviews that double-check that these guidelines have been followed correctly. These code reviews can be either manual or automated using technologies such as static application security testing (SAST).

That said, modern application developers can't be concerned only with the code they write, because the vast majority of modern applications aren't written from scratch. Instead, developers rely on existing functionality, usually provided by free open source components to deliver new features and therefore value to the organisation as quickly as possible. In fact, 90%+ of modern deployed applications are made of these open-source components. These open-source components are usually checked using Software Composition Analysis (SCA) tools.

Secure coding guidelines, in this case, may include:

- Using parameterized, read-only SQL queries to read data from the database and minimise chances that anyone can ever commandeer these queries for nefarious purposes
- Validating user inputs before processing data contained in them
- Sanitising any data that's being sent back out to the user from the database
- Checking open source libraries for vulnerabilities before using them

Phase 4: Verification

The Verification phase is where applications go through a thorough testing cycle to ensure they meet the original design & requirements. This is also a great place to introduce automated security testing using various technologies. The application is not deployed unless these tests pass. This phase often includes automated tools like CI/CD pipelines to control verification and release.

Verification at this phase may include:

- Automated tests that express the critical paths of your application
- Automated execution of application unit tests that verify the correctness of the underlying application

Phase 5: Maintenance and Evolution

The story doesn't end once the application is released. In fact, vulnerabilities that slipped through the cracks may be found in the application long after it's been released. These vulnerabilities may be in the code developers wrote, but are increasingly found in the underlying open-source components that comprise an application. This leads to an increase in the number of "zero-days"—previously unknown vulnerabilities that are discovered in production by the application's maintainers.

These vulnerabilities then need to be patched by the development team, a process that may in some cases require significant rewrites of application functionality. Vulnerabilities at this stage may also come from other sources, such as external penetration tests conducted by ethical hackers or submissions from the public through what's known as "bug bounty" programs. Addressing these types of production issues must be planned for and accommodated in future releases.

Secure SDLC benefits

As Secure Software Development Lifecycle integrates security tightly into all phases of the lifecycle there are benefits throughout the lifecycle, making security everybody's responsibility and enabling software development that is secure from its inception. Some of the biggest benefits are as follows:

- **Reduced Costs:** Thanks to early identification of security concerns allowing the embedding of controls in parallel. No more patching post-deployment.
- **Security-First:** Secure SDLC builds security-focussed cultures, creating a working environment where security comes first, and everyone's eyes are on it. Improvements happen across the organisation.
- **Development Strategy:** Defining security criteria from the outset improves technology strategy, making all team members aware of the security criteria of the product, and ensuring developer security throughout the lifecycle.
- **Better Security:** Once Secure SDLC processes are embedded, security posture improves across the whole organisation. Organisations that are security aware reduce their risk of cyberattack significantly.

Secure SDLC Best Practices

- **Culture:** Establish a culture where security is paramount. Identify key security concerns at project kick-off and build security into the code you develop from the beginning. Extend that security-first mindset to include dependencies, deployment tools, and infrastructure, protecting every link in the chain.
- **Standardisation:** Create a consistent Secure SDLC development roadmap, facilitating continuous improvement with embedded security. Create requirements that mandate security best practices, as well as tooling to help developers adhere to the process. Responses to security vulnerabilities should also be standardised, enabling consistency.

- **Testing:** Test regularly using static analysis security testing (SAST), shift left to start testing as soon as possible, and use threat modelling to keep your security position up to date as threats evolve. This ensures that code remains secure throughout the lifecycle by identifying deviations from accepted practices.
- **Penetration Testing:** While Secure Software Development Lifecycle promotes testing throughout the lifecycle, it does not mean an end for penetration testing. With Secure SDLC promoting testing throughout the lifecycle, penetration testing is often conducted later but remains the benchmark for risk management and proactive security.
- **Document and manage:** Security vulnerabilities identified during the development lifecycle must be documented, and remediation managed. These vulnerabilities may be discovered at any time with continuous monitoring and must be reacted to in a timely manner to prevent the risk profile and remediation costs from increasing.

Case Study: Software Development Life Cycle (SDLC) in Health Care:

Introduction:

This case study examines the implementation of the Software Development Life Cycle (SDLC) in the context of a healthcare organisation called "HealthTech Solutions." The organisation specialises in developing and maintaining various healthcare software applications, including Electronic Health Records (EHRs), Medical Billing Systems, and Patient Management Systems. This case study will focus on the development of a new EHR system, aiming to improve patient care, data accuracy, and overall efficiency within healthcare facilities.

Project Initiation and Planning:

In the initial phase, HealthTech Solutions identified the need for a modern and comprehensive EHR system to replace the outdated and fragmented systems being used by various healthcare providers. A project team was formed, including stakeholders from different departments such as physicians, nurses, IT specialists, and administrators. The team collaborated to define the project scope, objectives, and requirements.

Requirement Gathering and Analysis:

HealthTech Solutions conducted extensive consultations with various healthcare providers to gather detailed requirements for the EHR system. The team documented functional and non-functional requirements, ensuring compliance with relevant healthcare regulations and data security standards like HIPAA (Health Insurance Portability and Accountability Act). The team identified essential features such as patient demographics, medical history, lab results, prescription management, and appointment scheduling.

Design:

In the design phase, the development team, along with UX/UI specialists, translated the gathered requirements into technical specifications and a user-friendly interface. The team decided to follow an agile development approach to ensure regular feedback from end-users and iterative improvements.

Implementation:

The development team started building the EHR system in sprints, focusing on modular components and ensuring code quality and security. To address data privacy concerns, the team implemented robust encryption mechanisms and access control mechanisms. Continuous integration and automated testing were adopted to ensure stable releases and quick identification of issues.

Testing:

HealthTech Solutions performed rigorous testing throughout the development process. Unit tests, integration tests, and user acceptance tests were conducted to validate the system's functionality, interoperability, and usability. The testing phase included simulated scenarios to ensure the system could handle peak loads and potential data breaches.

Deployment:

After successful testing, the EHR system was deployed in a phased manner, starting with a limited set of healthcare facilities. This allowed for a smoother rollout and the identification of any unforeseen issues in real-world settings. Comprehensive training and user documentation were provided to healthcare staff to facilitate a smooth transition.

Maintenance and Support:

Once the EHR system was in use, HealthTech Solutions continued to monitor its performance, addressing any bugs or user feedback promptly. Regular updates and enhancements were released to improve the system's efficiency and incorporate new functionalities.

By adopting a structured Software Development Life Cycle, HealthTech Solutions successfully developed and deployed a robust Electronic Health Records (EHR) system. The new system improved patient care, streamlined healthcare processes, and enhanced data accuracy while complying with stringent healthcare regulations. The use of an iterative development approach ensured that the EHR system evolved with changing healthcare needs and technological advancements, ensuring its sustainability and long-term success.

Conclusion:

Studying case studies on Secure SDLC provides valuable real-world insights, fostering a deeper understanding of effective security practices in software development. These lessons serve as crucial guidelines for building robust and resilient applications in the face of evolving cyber threats.