Software Security through SDLC

Goutam Kumar Saha

gksaha@ieee.org

Software Development Life Cycle (SDLC) practices aim to improve software quality, reliability, and fault-tolerance. At any point of SDLC, developers are not completely sure whether the software is secure or not. In order to develop secured software, various security practices are being added to software engineering life cycle processes. The added security practices like penetration testing, and design principles like enforcing least privilege to the software activities and practices such as requirements engineering, modeling and model-based testing in each phase of the traditional software development life cycle (SDLC). Security is considered from the early stages of the software life cycle. All aspects of the SDLC can be directly controlled in developing custom software. Requirements specification considers basic system-level risk assessment on business assets, threats, risks likelihood and business impacts thereof. The context for the security aspects of software architecture and design is to be defined. Analysis to evaluate how the software addresses system risks is to be carried out and to suggest migration strategies and to identify additional risks to be added by software architecture. This is applied iteratively throughout the development life cycle in order to refine risk understanding. Several threat modeling methodologies can be used. Systems Security Engineering Capability Maturity Model (SSE-CMM) or an International Standard (ISO/IEC 21827) now, enables a system development organization to add security practices into their systems engineering (SE) CMM process.

Security relevant Quality Assurance throughout SDLC:-

(a) ***Requirements*** (Identification of down time acceptable levels and of data loss for Security requirements).

(b) ***Design*** (Identification and provision of countermeasures to vulnerabilities Design in order to incorporate activities related to forensics and disaster recovery and testability for both).

(c) ***Implementation*** (Automating nightly scans of code, vulnerability scans of application and database, and network & configuration scans; Documentation and use of manual security test methods).

(d) ***Testing*** (Penetration testing, black box testing to functional, compatibility, regression testing).

(e) ***Deployment*** (Security training to system administration, support staff and Help-Desk; Identifying security policy issues; Establishing schedule and procedures for system & Data backups & Disaster recovery).

(f) ***Operations & Maintenance*** (Repeating routine security reviews and vulnerability scans and Secure change control).

(g) ***Decommissioning*** (Media sanitization and appropriate disposal of hardware and software).

Eight Best Practices for Software Security:

(a) ***Static Analysis or Review of source code*** (tools to detect common vulnerabilities).

(b) ***Risk analysis of architecture & Design*** (assumptions documentation, possible attacks identification, uncovering along with ranking architectural flaws to mitigate, tracking recurrent risk, monitoring and ongoing analysis throughout the life cycle).

(c) ***Penetration Testing*** (black-box tests selection & implementation by tools for automated application security and penetration testing).

(d) ***Risk – based security testing*** (security functionality testing & risk based security testing of the software as a whole with attack pattern- based test scenarios).

(e) ***Building abuse cases*** (under attack system's behavior description- specifying areas and software- based system's components to be protected for how long from which threats).

(f) ***Security requirements specification*** (overt functional security using applied cryptography and revealed security properties out of abuse cases & patterns of attack).

(g) ***Security operations*** (monitoring the field deployed software system's behavior under attacks and software exploitations, and cycling back the learnt knowledge by monitoring attacks & exploits into other practices).

(h) ***External Analysis*** (outside analysts perform independent security reviews, assessments, and / or software's design and implementation tests.

As software security is a dynamic property and this is an emerging ongoing applied research topic towards developing software with three interrelated properties of dependability, trustworthiness and resilience.

Further Readings:

John D. McGregor "Secure Software." Journal of Object Technology, May 2005.

Mark Willoughby, "Quality Software Means More Secure Software", Computerworld March 2004.

Karen Mercedes Goertzel, et al. "Software Security Assurance," SOAR, July 2007.

Goutam Kumar Saha, "Understanding Dependable Computing Concepts," ACM Ubiquity, 8(44), November 2007.

Goutam Kumar Saha, "Security and Fault Tolerance – a CMap," IEEE Reliability Society Newsletter, 56(2), June 2010.

Goutam Kumar Saha, "Understanding Software Testing Concepts," ACM Ubiquity, 9(6), February 2008.