A Critical Analysis on Security Aspects of Software Development Lifecycle

Arghya Kusum Das
Department of Computer Science Engineering
Techno India College of Technology

Sandip Rakshit
School of Computing Science
Kaziranga University

ABSTRACT

This paper focuses on the security aspects of software. It analyses the various loopholes that can exist in the development of a software or the various damages that can be incorporated by malicious users, and also the remedies that when carefully undertaken can remove the vulnerabilities. This is an overview or study of security problems of different nature and the proper and systematic tackling methodology adopted to eradicate them and thereby also add value to its quality.

General Terms

Software Development Lifecycle. Software Security, Software Threat. Software Risk, Software Vulnerability

Keywords

Software Security, Software Threat. Software Risk

1. INTRODUCTION

Software usually caters to some specific requirement of the user implementation. However in today's world, where pace of change is rapid, it may not do so at all times. At all times it may not function or produce results as per the user's specifications. This may be due to some maintenance needs of the software that must be adhered from time to time. It might also be due to corruption by malicious users or destructive programmers. Hence, mere development of software alone is not sufficient. The overall development should be done keeping in mind the different faults or vulnerabilities that it might be exposed to or incorporated externally. This paper in short does an overall study of the different types of security faults that the software might face. Also to encounter, the different steps that needs to be undertaken to overcome or avoid those defects are also discussed.

2. PROPERTIES OF SECURE SOFTWARE

For any software to be secure it must adhere to five main properties ,three primary properties like *Confidentiality, Integrity and Availability* and additionally two secondary properties associated with human users like *Accountability and Non-repudiation* [1] .*Availability* states that the software must be accessible and operational to the intended authorized users whenever they want to access and use. *Integrity* states that the software must be protected from changes made by unauthorized person in an improper way which is also called as sub-version .Sub-version is achieved by unauthorized changes by authorized and unauthorized elements like overwriting, corruption, tampering, and destruction, insertion of destructive logic or simple deletion. Integrity of the software should be maintained in development and execution. *Confidentiality* states that the content, characteristics or behavior and existence should be hidden and obscured from unauthorized entities, most often prevent from learning about them. These primary properties are system-centric. However, additionally there are main two properties that are user-centric as it is related and dependent on human users. One such is *Accountability,* which requires all security relevant actions of the user in the software must be tracked and recorded in some log files and may be referred by an unauthorized user or hacker in future. Finally, *Non-Repudiation* is the ability to prevent users from denying or disapproving responsibility for actions it has performed like send or receive data, message or e-mail. The software would be consistent, stable and could be stated as threat-proof, if the above five mentioned properties are reserved and maintained

3. SCOPE OF SOFTWARE VULNERABILITIES & RISKS

The software that is produced as a product by the developers may not behave exactly in an expected manner giving proper and desired output to the user at all times after it is deployed. The software can behave in an irrational manner thereby giving undesired output at any stage after it is deployed .The abnormal behavior needs to be anticipated using case studies . This is due to the different cases in which the input to the program varies. The software should be checked for its functionality using extreme values. Boundary Value Analysis is an important aspect of functionality testing. Hence security engineer should create use cases to reduce the mis-use cases i.e. the abnormal behavior. This uneven behavior might be due to some fault, defect or bug in the software that might be created or externally injected or trapped [2]. The domain of these defects is vast. Defects refer to implementation errors and design errors of the developed software. A bug is usually a low- level implementation software error that can be captured by code-analysis of the external environment. A flaw is a defect into a much deeper level. Risk is the probability that a defect bug or flaw that may exist in the system that may affect the proper functioning of the software or make it prone to failure. The discussed fault, defect or bug primarily arises due to various reasons like buffer overflows, unauthorized access, malformed input, symbolic links, pathnames used, resource leaks format bugs and other miscellaneous causes. These unwanted elements might exist or arise in the underlying operating system, programming language used, network protocols ,size limits of variables used, access specifier,or cryptography also. Let us analyze this one by one by considering the common major causes [3].

Table I Primary causes of Software vulnerabilities

Serial No.	Cause	Occurrence (%)
1	Buffer Overflow	19
2	Unauthorized Access	16
3	Malformed Input	16
4	Symbolic Links	11
5	Pathnames	10
6	Resource Leak	6
7	Format Bugs	6
8	Others	16

3.1 Operating System

To start with, the software might be vulnerable depending on the nature or type of operating system used [4]. For example, there may be buffer overflows in Stack, Heap, Null Pointer, deadlock over resources. A buffer overflow may be caused when the program writes past the end of a buffer, resulting in corruption of adjacent memory contents. In some instances, this may result in overwriting the contents of the stack or heap in ways that allow an attacker to subvert the normal operation of the system and, ultimately, take over the flow of control from the program. At first, the attacker sends valid requests that result in allocating many chunks of memory based on the size of his input. Then by knowing that some memory once allocated can never be freed. Hence hard leaks occur, when the program forgets to free memory that was acquired during life-time. Defects like integer and buffer overflows, use of previously freed pointers, and use of scalars (for example, integers) that are not properly bounds-checked (sanitized) before being used as array or pointer indexes, loop boundaries, or function arguments are possible primitives for code execution. The Linux kernel is prone to local privilege-escalation vulnerability. Local attackers can exploit this issue to gain elevated privileges on affected computers or to cause a denial-of-service condition.

3.2 Database Level

Again depending on the programming language used there may be undesired conditions like exceptions encountered. The choice of the data-type and subsequently data size may also lead to errors like invalid data type. There may be vulnerabilities caused in the database level by SQL injections [5]. Using the SQL injections the hacker or programmer with destructive mentality may give as input some expert SQL commands which would make the SELECT statement use-less and not working specifically for a single authorized user , but may work universally i.e. though working does not serve the purpose for which it is written. For example let us consider a case with a simple login page where a authorised user would enter his username and password combination to enter his/her secure area to view his/her personal details or upload his comments in a common discussion forum. When this authorised user submits his details, an SQL query is generated from these details and gets submitted to the database for verification. If it is found to be valid, the user is allowed access. In other words, the web application namely the controller which may be a servlet or a jsp file or any other similar page that controls the login page will communicate this with the database through a sequence of planned commands so as to verify the username and password combination. On verification, the authorised user is granted appropriate access. Now, consider the query

```
SELECT * FROM users WHERE name = '' OR '1'='1';
```

As the second part i.e. after the OR clause is always true, hence the unauthorised-user may actually pass into the verification page and might get the unauthorised access of the authorised user profile and other details.

3.3 Network Level

Similarly, in the network level, privacy may be compromised as loop-holes might occur in awarding privileges. Pseudo anonymity in the internet is not secure and might be compromised. Let us think, in certain cases we want to visit certain sites for our reference or work, but at the same time we do not want our identities to get revealed. But as the IP address can be tracked hence our physical identity comprising of our locations gets revealed. In cryptography, management of keys is a sensitive area which if misused can cause a breakthrough in the system. For consideration, the Open SSL is prone to security by-pass attack, denial-of-service attack because it fails to properly process certain maliciously crafted S/MIME messages. Also while communication between parties, non-repudiation breakthrough can occur if technologies such as digital signatures are not used.

3.4 Application Level or Web-Level

XSS stands for cross-site scripting,is a type of vulnerability that is injection of active scripting data into scripting-enabled application screens.XSS targets script interpreting web-clients like web-browser, escalation of user-rights, code injection and client hijacking .Sometimes there might be security based vulnerabilities in the webmail, which tries to overwrite the security configuration settings of the user or manipulate the custom settings and preferences of the user. Again, PHPExcel is prone to information-disclosure vulnerability.

4. REMEDIES OR APPROACH ADOPTED

As it can be seen that the software developed might be exposed to a variety of security threats of different domain, hence to protect the software against these vulnerabilities software development following any of the classical model like waterfall, spiral or prototyping itself is not enough. Development of the model in these cases should be creative in anticipating the threats encountered and thereby taking appropriate steps at the different phases of the development lifecycle. The cost of corrective measures taken in the lifecycle of software development varies in direct proportional to the phase of its development; however the growth rate is different. This means that the cost of correction or other adoptive remedial measures is far less in the earlier stages of software development in comparison to the later stages of error detection and subsequent correction as shown in Figure 1[6].

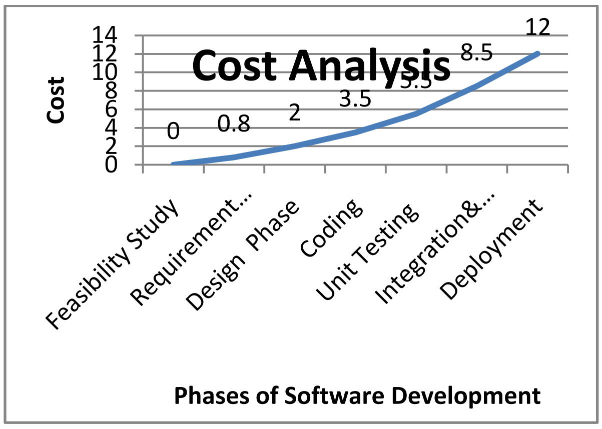

Fig.1: Phases of Software Development

4.1 Threat Modeling

Sometimes it is difficult for analysis tools to identify potential insecure areas of the software as the tools lack knowledge of the executing operation environment. In those cases, Threat Modeling is a useful tool to identify the risks and address the threats that has the potential to cause damage to the software. Here the system's data flow is analyzed to check for security loopholes. It identifies before the level of source-code implementation. Threat modeling tries to figure out the insecure business logic or work-flow .Thus, it is very beneficial to use Threat Modeling early in the life-cycle to optimize its effect [5].There are different approaches to the task of categorize threats which are as follows:-

4.1.1 STRIDE:-

Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service &Elevation of privilege. With using STRIDE, threat reduction tables can be used to figure out mechanisms to reduce threats.

4.1.2 MisUse Cases

It helps to understand how attackers might attack a system, and it is derived from the system requirement and reveals how the protective steps can be sidelined and security can be breached.

4.1.3 Threat Library

It is a template or library that makes threat identification accessible to non-security users. As the threat is identified it is checked and reduced or eliminated using the Common Vulnerability Scoring System (CVSS v2). There are tools that are useful in automated analysis of designs and relevant reduction steps, issue-trapping integration and communication related to process.

4.2 Security based Model of SDLC

Security based software development life-cycle is a defined methodology of software development where focus is on security of the software product created. Initially two security based models of software development were proposed. The first was Microsoft SDL as part of its Trustworthy Computing Initiative and the other was McGraw SDL [7]. Security was a major concern in both the SDLs in the requirement engineering phase. Microsoft promoted the concept of using a separate security assessment team to engineer and evaluate the security of its products. It is the responsibility of the software development team to identify all functional requirements including the security functional requirements. Each team has a security engineer who reviews the product plan, functional requirements and determines security milestones and exit criteria. These requirements are well documented. On the other hand, McGraw focussed on the use of abuse case in requirement engineering phase.

4.3 Security Enhanced SDLC Methodologies

An SDLC methodology identifies the set of activities to be performed at each phase of SDLC, and in some cases also what artefacts should be produced as a result. There are mainly two defined methodology, CLASP and TSP which are discussed as follows:-

4.3.1 CLASP

The Comprehensive, Lightweight Application Security Process (CLASP) is a collection of methods and practices that can be used collectively to identify and take suitable actions for appropriate application security concerns before any sourcecode is written for the system. CLASP is the first defined life cycle process with the specific aim of enhancing the security (versus safety, correctness, or high-quality) of the early stages of the software development life cycle. As a formal process emphasizing accuracy and quality assurance, CLASP shares objective traits native to more industry-driven CMM based life cycle process models. CLASP includes instructions, guidance, and checklists, for activities that comprise its structured process. Thirty (30) specific activities are expressed in CLASP which can be used to increase security awareness across the development team. These activities are assigned to the typical roles found throughout the life cycle comprising of both owners and participants. CLASP assigns responsibility and suggests accountability for each activity, and create two different paths: one that supports new system development using an iterative, or "spiral", methodology, and one that supports maintenance/enhancement of legacy systems with the focus on management of the current development effort. Both roadmaps include consistent testing and analysis of the application's security posture through any upgrades or enhancements. CLASP is available as a plug-in to the Rational Unified Process (RUP) development methodology. The CLASP plug-in to RUP is available free-of-charge but requires a RUP license to install.

4.3.2 TSP-Secure

Software Engineering Institute (SEI) and CERT/CC jointly developed the Team Software Process for Secure Software Development The aim of TSP-Secure is to reduce the vulnerabilities that can exist in the design and also to predict the probability of security concern areas in the delivered product. TSP-Secure provides methods for analyzing the defect type, design patterns for common vulnerabilities removal of threat prone areas in legacy systems.

4.4 Process based Model of SDLC

Capability Maturity Models are kinds of process model that provides guidelines to improve the process and evaluate the capability of operations. Currently there exist three CMMs that address security, the Capability Maturity Model Integration (CMMI), the integrated Capability Maturity Model (iCMM), and the Systems Security Engineering Capability Maturity Model (SSE-CMM) [8]. A common Safety and Security Assurance Application Area is currently under review for the iCMM and CMMI, along with a new process area for work environment, and the proposed goals and practices have been piloted for use. All of these CMMs are based on the Capability Maturity Model (CMM).

4.5 Design Based Secure Approach of SDLC

UMLSec an extended version of UML is a designing tool that is used for implementing security in the software lifecycle [9] . Here, possible security threats or loopholes are checked and subsequent actions are taken in the design level.UML Sec is a reveals security related information with the specification diagram such as various interaction diagrams and deployment diagrams. UMLSec is formed as a UML profile which mainly uses stereotypes, tagged values and constraints. The profile has some related fixed set of stereotypes. Every stereotypes have variables where tag is the real life object of the variable. Once the stereotype is formed, tag, constraints and threat specification can be chalked out. Following the model, security requirements can be found and threat specification steps needed can be specified. UMLSec also supports cryptographic fundamentals like encryption, decryption, digital signature. Apart from UMLSec , there are other types of UML, like CORAL UML that introduces a meta model by defining stereotypes. The focus of the CORAS UML profile is the modeling of threats, such as buffer overflow exploits and remedies undertaken. Another version of UML, SecureUML, is a UML-based modeling language is used for expressing Role Based Access Control (RBAC) and authorization constraints in the overall design of software systems.

4.6 Defined Standards for Software Project Lifecycle Process

IEEE 1074-2006 supports proper implementation of levels of security controls into software and systems. The aim of both IEEE Std. 1074-1997 and ISO/IEC 12207 is to chalk out a quality-driven SDLC process, for that reason, neither standard contains specific security guidance, though ISO/IEC 12207 does suggest the need for security activities, or provides references to security-specific standards . IEEE 1074-2006 is an improvement on ISO/IEC 12207 in particular in that it generates the security information needed to document security risks and solutions throughout the SDLC.

5. CONCLUSION & FUTURE SCOPE

This paper is a study of the different possible innumerous vulnerabilities that may exist or arise in the different levels of software deployment. These may be unintentional or intentionally trapped by destructive programmers and hackers that may be able to trap and capture the loop-holes as discussed in section 3. Also, the different measures that can be taken to prevent these unwanted situations are also discussed in section 4. Though discussed the section 4 may not be sufficient alone, and hence there is a huge scope of work in implementing the security aspects either in the different phases of software development lifecycle or in the different levels of software deployment and maintenance once it is deployed and made online .Security aspects can also be embedded in the existing software development process models to make it more complete and make it a comprehensive attractive package for customers buying the software.

6. ACKNOWLEDGMENT

We would like to take this opportunity to thank Professor Pranab Ray of School of VLSI,IIEST, Shibpur and also the staff members from my institution without whom this work would not had progressed and would have been incomplete.

7. REFERENCES

[1] Karen Mercedes Goertzel, Theodore Winograd, Holly Lynne McKinley,Patrick Holley, Booz Allen Hamilton , “Security in Software Lifecycle Making Software Development Processes and Software Produced by Them- More Secure”, Department of Homeland Security , Draft Version 1.2- August 2006 .

[2] Jayaram K R andAditya P Mathur ,” Software Engineering For Secure Software - State Of The Art: A Survey” , CERIAS Tech Report 2005-67

[3] Ansar-Ul-Haque Yasar ,Davy Preuveneers ,Yolande Berbers ,Ghasan Bhatti, ”Best Practices for Software Security : An Overview”, In Proceedings of the 12th IEEE International Multitopic Conference, December 2008.

[4] Malik Imran Daud,” Secure Software Development Model: A Guide for Secure Software Life Cycle”, In Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol. I, Hong Kong.

[5] Mark Belk, Matt Coles, Cassio Goldschmidt, Michael Howard, Kyle Randolph,” Fundamental Practises for Secure Software Development”, February 2011.

[6] J. Christopher Westland “ The Cost of Errors in Software Development :Evidence From Industry “,The Journal of Systems and Software, 62,2002,pp. 1-9.

[7] Hossein Keramati, Seyed-Hassan Mirian-Hosseinabadi ,” Integrating Software Development Security Activities with Agile Methodologies “ .

[8] Noopur Davis ,”Secure Software Development Life Cycle Processes: A Technology Scouting Report”, December 2005.

[9] Golnaz Elahi ,” Security Requirements Engineering: State of the Art and Practice and Challenges”