ICIC Express Letters ICIC International ©2018 ISSN 1881-803X
Volume **12**, Number **11**, November **2018** pp. 1155–1162

MAJOR SOFTWARE SECURITY RISKS AT DESIGN PHASE

Jasleen Kaur*, Alka and Raees Ahmad Khan

Department of Information Technology
Babasaheb Bhimrao Ambedkar University (A Central University)
Vidya Vihar, Raebareli Road, Lucknow 226025, India
*Corresponding author: jasleenkaur.lmp@gmail.com

Received April 2018; accepted July 2018

Abstract. *In the present world of big data where each individual is highly concerned about the security of their data, it becomes the basic responsibility of both academicians and developers to manage the security issues in an efficient manner. In order to effectively address security, risk management needs to be considered as the key element. Mostly, security is believed to be an add-on and not the fundamental issue during software development. The authors have laid stress on considering security at initial developmental phases. This will prove to be beneficial in development of software that will itself be able to fight against threats and not rely on any external application security program. In the research work, the authors have identified some risks that are most likely to be introduced during design phase from Common Weaknesses Enumeration (CWE) list. CWE is sponsored by the National Cybersecurity FFRDC, which is owned by The MITRE Corporation.*

Keywords: Software security, Security risks, Design phase, Software development life cycle, Object-oriented design properties

1. **Introduction.** In the present world, almost all of the services are being provided via computers. There has been a significant increase in software deployment in almost every field. This introduces an urgent need to address security issues as security failure may lead to disastrous effects on human lives [1]. G. McGraw has already stated that security is not the thing that can be sprayed onto any software after its development rather it must be analyzed throughout the development phases [2]. This will help to build software that will actually be able to defend itself from attacks despite being dependent upon any application security software (say, *antivirus*) for its protection against threats. The basic cause of the maximum of the security breaches is the presence of loopholes in the end product. The early detection and correction of these ambiguities may help reduce the occurrence of such attacks. The basic idea behind this research work is to identify the security risks as early as possible and nib them in the bud itself. The security risks, if removed from the design phase, will prove to be very helpful to build a secure and efficient software. The design phase generally aims at prevention of introduction of defects [3]. So, in order to reduce the occurrence of security violations, it becomes indispensable to address the security risks that exist at the design phase of software development life cycle. The identification of the security risks that can be rectified at the initial phases will help 'build' security into the software.

Today, most of the researchers are emphasizing the idea of addressing the security issues at the initial phases of software development life cycle. The early detection and correction of security risks is said to help deal with the prevalent security aspects in software development [4]. P. T. Devanbu and S. Stubblebine [5] have stressed on consideration of security issues at every phase of development life cycle. The authors have also outlined

DOI: 10.24507/icicel.12.11.1155

the idea of refining the requirement and design processes so as to shift the focus on initial developmental levels. Baker et al. [6] have dragged the focus towards the lack of valid methodology to quantify the effectiveness of the security measures. According to the authors, it is not the scarcity of security methodologies that hinders the development of secure software but the absence of proper quantification tools.

D. M. Mehta [7] has highlighted the idea of integrating security in the development process. The author has also stated that the only thing that can help in development of secure software is modifying the development life cycle. S. Gupta [8] has insisted on the application of risk management strategies in the early stages of software development. The author has also proclaimed that late risk management indirectly poses greater threats to secure software development. Steps such as identification of threats, vulnerabilities and determining the appropriate risk mitigation strategies at the design phase have also been proposed by the researcher. B. Gray [9] has elaborated the role of a security analyst. The author has talked about the importance of phase-wise implementation of security policies and risk management during software development. A researcher has also provided a design phase security checklist for developing a web application [10]. The checklist emphasizes on the security issues based on the principle of 'Defense in Depth'. Through the literature review it is evident that security, if considered at initial development phases may result in comparatively more secure software. On the other hand, it is also obvious that none of the researcher has identified or analyzed the security risks existing at the early developmental phases. So, this research work may prove to be advantageous and favorable towards secure software development.

The rest of the paper is organized as follows. Section 2 presents the need for security risk identification at early stages and further in Section 3, the major security risks existing at the design phase have been identified. Section 4 discusses the relation between security risk and security factors. Finally, the research paper concludes in Section 5.

2. Need for Design Level Security Risk Identification.

Security is widely known to be a combination of two parts, viz., effective risk management and application of proper countermeasures [25]. Risk assessment is widely accepted as an integral part of risk management process. The risk assessment process is a complex procedure which consists of the following sub-steps: identification of various risks; assessment of the vulnerabilities; establishment of threats and their countermeasures; preparation of corrective action plan; review and monitoring [26]. As the first step itself is the identification of the risks, it becomes a prerequisite to pin down them. Also, the basic aim of risk assessment is to provide apt security levels of a system by ranking the risk on the basis of severity of its impact.

Additionally, the recognition of different security risks at design phase will help avoid the loopholes that may pose a threat to security of the system in the future. If the design itself is prepared in such a way that the security related risks are evaluated, then it may help in reduction of time and money that is spent on application security software. Detecting and rectifying bugs after development is found to be 100 times critical as compared to considering them at the design phase [11]. Therefore, it is suggested to address the security related risks at early stages of software development.

3. Major Security Risks at Design Phase.

The researchers have selected the critical risks based on the related security factor. Addressing security factors such as confidentiality, access control, authentication, and integrity, has become a pre-requisite for secure software development. Especially today, when each and every individual is primarily concerned about the security of his data, it becomes the prime responsibility of the software developers to effectively address them. Therefore, in this proposed work, the authors have filtered the security risks that may penetrate into the software at design phase from Common Weaknesses Enumeration (CWE) list [12]. The CWE is a community that facilitates the secure software development by providing a list of all possible weaknesses that may occur in any software. It serves as a security tool by providing a standard for identification and mitigation of various software weaknesses. The major design-level security risks, as identified by the researchers, have been shown in Figure 1 and Table 1 shows the relation of the security risks with the security factors along with risk-definition.

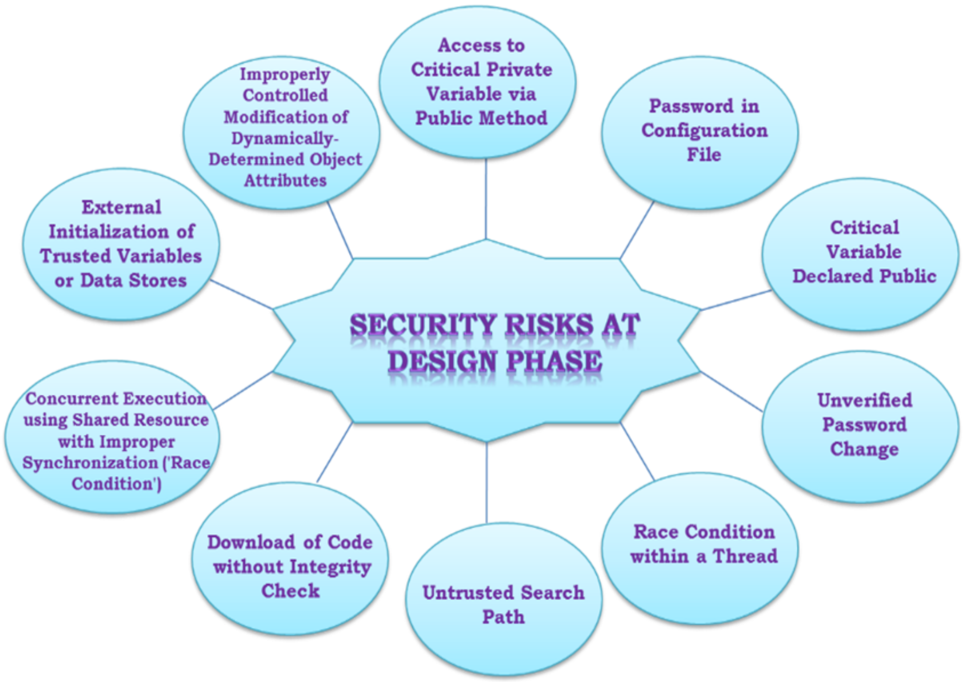
SECURITY RISKS AT DESIGN PHASE
Access to Critical Private Variable via Public Method
Password in Configuration File
Critical Variable Declared Public
Unverified Password Change
Race Condition within a Thread
Untrusted Search Path
Download of Code without Integrity Check
Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
External Initialization of Trusted Variables or Data Stores
Improperly Controlled Modification of Dynamically-Determined Object Attributes

Figure 1. Security risks at design phase

4. **Relation between Security Risk and Security Factors.** Risk refers to the potential for loss or damage when a threat exploits vulnerability [24]. The risks which pose danger to the security of the software can be categorized as 'security risks'. In this section, the authors have explained that why the identified risks are being termed as 'security risks' and the relation between the identified security risks and respective security factors (Table 1) has also been illustrated below.

4.1. **Access to Critical Private Variable via Public Method (ACPVPM).** There may be a possibility that any crucial attribute (variable) that has been declared as 'private', can be accessed through any public method. This may occur because of the non-compliance of proper methods for access control at design phase. Consequently, it may lead to the non-observance of the suppositions of other parts of the code. Also, if an attacker is able to access any private variable, then it may lead to leakage of any sensitive information. Consequently, it will affect the integrity of the code. For example:

```
private: int age;
public:
void UpdateAge (int age)
{
cout << "Update your age";
cin >> age;
}
```

In the above example, 'age' has been declared as private but it is being accessed from a public function for its updation.

Table 1. Security risks and related security factor

S. No.	Security risk at design phase	Definition	Related security factor
1	Access to Critical Private Variable via Public Method [12] **(ACPVPM)**	A public method that can read or modify a private variable is defined by the software [13].	Access control; Integrity
2	Password in Configuration File [12] **(PCF)**	Password is stored in the configuration file, thereby making it prone to be misused by any outsider [14].	Authentication; Access control
3	Critical Variable Declared Public [12] **(CVDP)**	Any critical variable/field is declared as public when intended security policy requires it to be private [15].	Confidentiality; Integrity
4	Unverified Password Change [12] **(UPC)**	No authentication mechanism is followed while setting a new password for a user [16].	Authentication; Access control
5	Race Condition within a Thread [12] **(RCT)**	If any resource is being used simultaneously, then there exists the possibility that resources may be used while invalid and consequently making the execution state undefined [17].	Integrity
6	Untrusted Search Path [12] **(USP)**	An externally-supplied search path is being used for critical resources that can point to resources that are not under the application's direct control [18].	Confidentiality; Integrity; Availability; Access control
7	Download of Code without Integrity Check [12] **(DCIC)**	An executable source code is downloaded from any remote location without checking the origin and integrity of the code [19].	Integrity; Confidentiality
8	Concurrent execution using shared resource with improper synchronization ('Race Condition') [12] **(RC)**	The program contains a code sequence that can run concurrently with other code, and the code sequence requires temporary, exclusive access to a shared resource, but a timing window exists in which the shared resource can be modified by another code sequence that is operating concurrently [20].	Integrity; Confidentiality
9	External Initialization of Trusted Variables or data stores [12] **(EITV)**	The software initializes critical internal variables or data stores using inputs that can be modified by suspicious actors [21].	Integrity
10	Improperly Controlled Modification of Dynamically-determined object attributes [12] **(ICMD)**	If the object contains attributes that were only intended for internal use, then their unexpected modification could lead to vulnerability [22].	Integrity

4.2. **Password in Configuration File (PCF).** The configuration files are the files that are used by the developers to manipulate the settings without recompilation and by administrators to set policies regarding various applications running on a computer system [23]. The designer should be very careful that the software does not store any kind of password in the configuration files. This may lead to a security breach where any non-legitimate user gains the password thereby affecting authentication. This risk also poses danger to access control if the password gets changed by the attacker.

4.3. **Critical Variable Declared Public (CVDP).** The public declaration of any critical variable/field that should necessarily be declared private, as per security policies (say, *password*) is referred as a security risk. It may act as a threat to confidentiality when the information attached to this kind of attribute is read by any non-legitimate user and to integrity if that information is further modified by the attacker. For example:

```
class login
{
public:
credentials(char *UID, char *password)
        {
        if((strlen(UID)>MaxLength-1)||(strlen(password)>MaxPassword-1))
cout<< "Invalid UID or password";
        else {
              strcpy(this→UID, UID);
        strcpy(this→password, password);
        }
        int authenticate(char *UID, char *password)
              if((strlen(UID)>MaxLength-1)||(strlen(password)>MaxPassword-1))
            cout<< "Invalid UID or password";
              else {
        strcmp(this→UID, UID);
        strcmp(this→password, password);
                    return 0;
                    else
                    return 1;
                    }
        char UID[MaxLength];
        char password[MaxPassword];
        }
}
```

In the given example, the member variables UID and password are declared public and therefore will allow access and modifications privilege to anyone with access to the object.

4.4. **Unverified Password Change (UPC).** The unverified modification of password may serve as a threat to authentication and access control. Authentication is believed to be overlooked if while setting the new password, the software does not demand previous password, or any other kind of authentication. The non-consideration of authentication factor consequently makes room for access control vulnerability. The password, if changed by non-legitimate user, may limit the valid user from accessing the information. While the software confirms the changed password twice, it should also check that the password is being changed by the authorized user or not.

4.5. **Race Condition within a Thread (RCT).** This condition is said to occur when two or more threads try to access a shared memory location and make an attempt to change it simultaneously. It may hinder the integrity of the application if any value being used currently is changed by any other piece of code at the same time. The mitigation scheme of this type of risk is believed to be 'locking'. Proper locking mechanism is assumed to help by preventing the usage of shared memory location at the same point of time. Usage of flags or signals may also prove to be beneficial in this case.

4.6. **Untrusted Search Path (USP).** This type of risk may occur when the application allows access to the critical sources through any externally-supplied search path. For example, if any search URL contains the primary key (say, *User ID*) and just altering the same allows the user to access another user's profile without permission. If the attacker tries to modify the content, the integrity of the system may also get damaged. The availability is put to risk if the attacker redirects the application to wrong files or compels the software to crash or hang. If the result of any query is directed towards a non-legitimate user, then the confidentiality is said to get affected. Therefore, this risk directly poses danger to confidentiality, availability, access control and integrity of the software.

4.7. **Download of Code without Integrity Check (DCIC).** There may be a possibility that the software allows the downloading of code from any remote location without checking the authenticity of the source. The pressure to reduce the time-to-market exists on the developers which consequently increases the chances of software updation through mobile code. Therefore, it should be the prime condition to make a check on the validity of the source. The avoidance in testing the legitimacy of the source may affect the confidentiality and integrity of the application. If the malicious code downloaded from any unknown source tends to change the working of the software, then integrity is put to risk and if the critical details get leaked, then confidentiality is believed to get hampered.

4.8. **Concurrent execution using shared resource with improper synchronization ('Race Condition') (RC).** The 'Race Condition' is said to occur when the application contains a piece of code that can run simultaneously with another code section. It should be strictly followed that no processes are able to share the same memory location except for the case that all of them only need to read the data from that particular location and none of them needs to update it. The security risk because of race condition occurs when the shared code is about the access of a critical resource. In such a case, it may be a possibility that the intruder accesses or overwrites a critical data, which in turn may affect integrity and confidentiality of the data source.

4.9. **External Initialization of Trusted Variables or data stores (EITV).** The developers need to strictly adhere to the fact that if any variable gets initialized externally, then its improper initialization may lead to an increase in security vulnerability of the software. The unexpected initialization may lead to an abrupt response from the software. Similarly, the blind input to the data stores may also introduce flaws. It may therefore be concluded that this security risk poses direct danger to integrity of the software system.

4.10. **Improperly Controlled Modification of Dynamically-determined object attributes (ICMD).** It may happen to be that the inputs to the attributes of any object are governed by an upstream component. This condition may lead to serious security vulnerabilities such as mass-assignment, and object-injection [22]. The possible methods to cope up with such kind of risk include the separation of object attributes as accessible and protected or association of an authentication code to the deserialized data at the time of storage.

5. **Conclusion and Future Work.** The old proverb, "Prevention is better than cure", very aptly describes the fundamental issue being emphasized by the authors. If the security issues are tackled at their budding phase, then it will greatly help in reduction of security violations. The proactive approach to development of secure software needs to be

prioritized. It is expected that if the detection of any loophole is done at the initial level, then it may consequently lead to more efficient and secure software. In the present world, where almost everything is being digitized, the use of object-oriented technology tends to increase automatically. The security aspect cannot be overlooked at the same time. So, in future if these security risks are linked to the object-oriented design properties, then it may prove to be very helpful for secure software development. Further, the quantification of relation of these risks with object-oriented design properties may provide the researchers with their accurate interdependence. The exact mutual dependability can further be utilized to develop a secure, efficient and reliable software.

Acknowledgement. This work is sponsored by UGC-MRP, New Delhi, India under F. No. 43-391/2014 (SR).

REFERENCES

[1] H. de Bruijn and M. Janssen, Building cybersecurity awareness: The need for evidence-based framing strategies, *Government Information Quarterly*, vol.34, no.1, pp.1-7, 2017.

[2] G. McGraw, *Software Security: Building Security In*, Addison-Wesley Professional, 2006.

[3] R. K. Choudhary and R. A. Khan, Testing software fault tolerance techniques: Future direction, *ACM SIGSOFT Software Engineering Notes*, vol.36, no.3, pp.1-5, 2011.

[4] *Software Integrity*, https://www.synopsys.com/blogs/software-security/secure-sdlc/, 2017.

[5] P. T. Devanbu and S. Stubblebine, Software engineering for security: A roadmap, *Proc. of the Conference on the Future of Software Engineering*, Limerick, Ireland, pp.227-239, 2000.

[6] W. Baker, L. Rees and P. Tippett, Necessary measures: Metric-driven information security risk assessment and decision making, *Communications of the ACM*, vol.50, no.10, pp.101-106, 2007.

[7] D. M. Mehta, *Effective Software Security Management*, OWASP, https://www.owasp.org/images/2/28/Effective_Software_Security_Management.pdf, 2017.

[8] S. Gupta, *A Proactive Approach to Information Security*, SANS Institute InfoSec Reading Room, https://www.sans.org/reading-room/whitepapers/securecode/proactive-approach-toinformation-security-1416, 2003.

[9] B. Gray, *The Role of the Security Analyst in the Systems Development Life Cycle*, SANS Institute InfoSec Reading Room, https://www.sans.org/reading-room/whitepapers/awareness/role-security-analyst-systems-development-life-cycle-1601, 2005.

[10] G. Z. Bayse, *A Security Checklist for Web Application Design*, SANS Institute InfoSec Reading Room, https://www.sans.org/reading-room/whitepapers/securecode/security-checklist-web-application-design-1389, 2004.

[11] B. Boehm and V. R. Basili, *Software Defect Reduction Top 10 List*, https://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf, 2017.

[12] *Weaknesses Introduced during Design*, https://cwe.mitre.org/data/definitions/701.html, 2017.

[13] *CWE-767: Access to Critical Private Variable via Public Method*, https://cwe.mitre.org/data/definitions/767.html, 2017.

[14] *CWE-260: Password in Configuration File*, https://cwe.mitre.org/data/definitions/260.html, 2017.

[15] *CWE-766: Critical Variable Declared Public*, https://cwe.mitre.org/data/definitions/766.html, 2017.

[16] *CWE-620: Unverified Password Change*, https://cwe.mitre.org/data/definitions/620.html, 2017.

[17] *CWE-366: Race Condition within a Thread*, https://cwe.mitre.org/data/definitions/366.html, 2017.

[18] *CWE-426: Untrusted Search Path*, https://cwe.mitre.org/data/definitions/426.html, 2017.

[19] *CWE-494: Download of Code without Integrity Check*, https://cwe.mitre.org/data/definitions/494.html, 2017.

[20] *CWE-362: Concurrent Execution Using Shared Resource with Improper Synchronization ('Race Condition')*, https://cwe.mitre.org/data/definitions/362.html, 2017.

[21] *CWE-454: External Initialization of Trusted Variables or Data Stores*, https://cwe.mitre.org/data/definitions/454.html, 2017.

[22] *CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes*, https://cwe.mitre.org/data/definitions/915.html, 2017.

[23] *Configuration Files*, https://msdn.microsoft.com/en-us/library/1xtk877y(v=vs.100).aspx, 2017.

[24] *IT Security Vulnerability vs Threat vs Risk: What's the Difference?*, http://www.bmc.com/blogs/security-vulnerability-vs-threat-vs-risk-whats-difference/, 2017.

[25] *Chapter 1 – Web Application Security Fundamentals*, https://msdn.microsoft.com/en-us/library/ff648636.aspx, 2017.

[26] *IT Risk Assessment*, https://www.happiestminds.com/whitepapers/IT-risk-assessment.pdf, 2017.