2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)
August 28-29, 2018; Shahid Rajaee Teacher Training University - Tehran, Iran

Software Security Analysis Based on the Principle of Defense-in-Depth

Ahmad Jalali
Malek Ashtar University of Technology
Department of Computer and Electrical Engineering
Tehran, Iran
a.jalali@mut.ac.ir

Mohammad Ali Hadavi
Malek Ashtar University of Technology
Department of Computer and Electrical Engineering
Tehran, Iran
hadavi@mut.ac.ir

***Abstract*— Defense in depth is a well-known secure design principle. Although the software security community acknowledge the importance of such a principle in developing secure systems, it has not been investigated enough to be the basis of security analysis of software systems. In this paper we analyze software security with respect to the defense-in-depth principle. We propose a model for security analysis in which the defense-in depth is quantitatively measured. The measurement capability lets developers choose security countermeasures in such a way that not only the security risks decrease but also the amount of defense-in-depth increases. We experimentally evaluate our model using a case study. The results show that adding security countermeasures to reduce security risks has different effects on security with respect to the defense-in-depth, and implementing a security countermeasure, while reducing the total risk, does not necessarily lead to an improved amount of the defense-in-depth.**

Keywords—security; security risks and countermeasures; defense-in-depth; security measurement

I. INTRODUCTION

Nowadays, software has become the first communication lane between information systems and its users. User's data has been uploaded on different systems while they wish to have access to their data anywhere and anytime. In such an environment, security is a great concern for users. To achieve better security, there are some secure design principles should be considered while developing systems. *Defense-in-depth* or *layered defense* is amongst the most important ones.

Defense-in-depth tries to employ security countermeasures in different layers to increase the whole system security. When we use security countermeasures in separate layers, if one layer fails, there would be other countermeasures to resist against threats or to decrease their risks. Integrating replicated - but not layered - security countermeasures make attacker to overcome all the countermeasures as a result of a successful attack. Moreover, the idea behind defense-in-depth addresses *a single point of failure* design anti-pattern in our system [1].

An example of defense-in-depth could be a system in which data is passing between clients and servers, e.g., a database server. Most companies keep attackers away from their system by using firewall. Then, they may think that firewall provides enough security for data being transferred to the database. With having in mind that the data is important, what would happen if someone could penetrate into the firewall? If data was encrypted, the attacker could not get any access to it unless he has the decryption key. In this scenario, data encryption is another protection layer, in addition to the firewall protection. However, the main remaining question is "how much is the system security with respect to the defense-in-depth principle?"

Measurement is the basis of science and engineering. If we wish to speak about the security of software, we should be able to measure it. Measuring security can help software developers who need to claim about the security of their product. It also helps organizations and end users to know about the security of their data stored and shared through the software. Finally, it helps security analysts to fairly judge concerning the security of a software application. However, existing works to this purpose, usually propose measurement models, which are not concrete enough to be practical for measuring the security of complex software applications. In addition, the metrics used for measuring have not been derived from well-accepted design principles such as the defense-in-depth, open design, and least privilege.

In this paper, we focus on the principle of defense-in-depth and propose a quantitative measurement method in which software security is analyzed with respect to the defense-in-depth principle. We examine our method on a real software whose risks are mitigated through security countermeasures, and show the effect of different countermeasures on the defense-in-depth quantity in the system.

The rest of this paper is organized as follow. In Section II we review related works. In Section III we introduce our proposed model. Section IV reports the results of our case study. Finally, the paper is summarized and concluded in Section V.

II. RELATED WORKS

There are some works related to measuring software security. In [2] six CVSS-based metrics [3] have been

978-1-5386-7582-3/18/$31.00 ©2018 IEEE

introduced. There is also a method to quantitatively measure these metrics. Each metric is given a grade between one and ten so that a vector of grades is generated which is used further for security measurement.

In [4] software security has been measured using SAN[1] models. This paper uses metrics such as "the probability of a successful attack" and "the amount of time to perform a successful attack" to measure software security. The authors map Markov modeled attacks on SAN models. Then, they implement the SAN models by a simulator called Mobius and calculate security metrics.

There are also some other works related to the defense-in-depth. Although there are several studies on either measuring security or improving defense-in-depth, few works have been reported for measuring the defense-in-depth. The concept of *layer* in the defense-in-depth is discussed in [7]. The authors use attack vectors derived from the attack tree to determine layers of the system. Then, after finding the layers, defense-in-depth is measured. In [5] a design solution for IPv4 and IPv6 networks has been proposed to improve the defense-in-depth. In [6] attack graph has been used for measuring. The authors first gather information about the network by probing it. Then, they draw an attack tree and perform Hypothetical patches to measure its effects on the system. At last, they propose a list of patches which are the best for the network concerning the defense-in-depth.

Rabiee *et al*. [8] propose a model to analyze and measure software security. Their model considers software threats and corresponding countermeasures to quantitatively compute software remaining risk. The model can also compute the defense-in-depth relying on the overlapping of countermeasures associated to a specific threat. Security in this study is measured based on the weakest link. That is, a security countermeasure is chosen so that decreases the risk of the weakest link in the software.

Our approach is aimed at measuring and analyzing security with respect to the defense-in-depth, while in [8] software weakest link is the basis of security measurement. We have also improved the model in [8] in terms of defining the concept of *layer* for security countermeasures.

III. PROPOSED APPROCH

Our proposed model, describe a software system S as a triple (T, C, G) where:

1. $T = \{t_1, t_2, \dots, t_n\}$ is the set of existing threats to our system. Every threat t_i is assigned p_i, which is the probability of occurring t_i and also $impact_i$, which is the severity of t_i's effect on the system when it occurs ($0 < impact_i \leq 1$).
2. $C = \{c_1, c_2, \dots, c_n\}$ is the set of countermeasures, where $c_i(cost_i, l_i)$ is a security countermeasure in the layer l_i with implementation costs $cost_i$. Cost of a countermeasure indicates the required effort to implement it and can be measured by person per day, the line number of code to be added, and etc.
3. G is a bipartite graph whose vertices are $T \cup C$ and whose edges connect countermeasures to threats. An edge e_i connects a vertex $c_i \in C$ to a threat $t_i \in T$, meaning that c_i is a countermeasure for t_i. e_i has a label f_i, indicating the percentage of t_i restrained by c_i ($0 < f_i \leq 1$). A countermeasure c_i can cover multiple threads with different coverage ratios (f is the coverage ratio). Vice versa, a threat t_i can be mitigated (partially or totally) by several countermeasures. Therefore, we model an edge in G by a triple (c, t, f) where c mitigate f ratio of t. the triple (c, t, f) indicates an edge in the graph. Therefore G is modelled by (C, T, E) where E is a set of triples. In Figure 1 we see a simple graphical view of our model

$$T = \{(p_1, impact_1), (p_2, impact_2), \dots, (p_m, impact_m)\}$$
$$C = \{(cost_1, l_1), (cost_2, l_2), \dots, (cost_n, l_n)\}$$

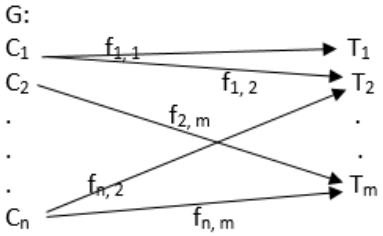

Figure 1- a brief picture of our input model

The triple (T, C, G) models the software and is considered as the system input. This helps us reaching a bright sight about the security of our system with respect to the concept of defense-in-depth. Using this model, in the first step we analyze our inputs. Then, in the second step, we measure the defense-in-depth. Finally, we propose a countermeasure to improve the defense-in-depth in our system. Figure 2 shows the general process of our approach.

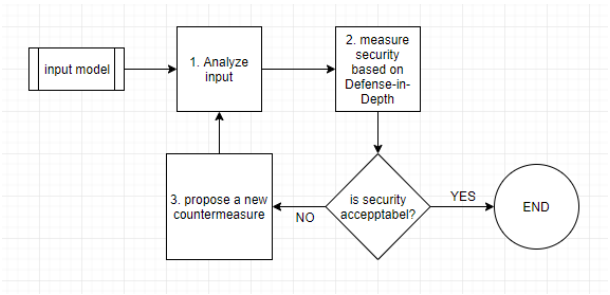

Figure 2- flow chart of the proposed approach

A. Defining layers

We review security mechanisms used in software systems in order to identify suitable layers in which countermeasures protect systems against threats. Eventually, we identified five layers of protection. That is, in our model l_i is a value in L where L = {*Physical, DataAccess, Business, Network, User*}.

[1] Stochastic Activity Networks

Clearly, the above set says that choosing the value l_i for a countermeasure c_i depends on the place of implementing c_i. If a countermeasure is implemented for physical resources of our system, such as hard disks and cables, countermeasure would be counted for *Physical* layer. Examples of such controls in *Physical* layer are storing encrypted or hashed data or configuring our physical devices safely. Countermeasures, which protects access to data (in database, for instance) are placed in *DataAccess* layer. Examples of such controls include preparing a safe medium to access our DBMS to stored data. The layer of *Business* contains security protections implemented in the business logic of our software system. For instance, designing a safe authentication system for our software or provide periodic patches for our software are countermeasures in the *Business* layer. *Network* layer countermeasures are those protect the system by securing the network in which our system is deployed. Using firewall or changing its policies by defining new rules in order to filter malicious traffic towards a system is considered as a *network* layer countermeasure. Putting an IDS in the network that the software is running is another example of a *network* countermeasure. Any human related countermeasures such as user training and awareness are among *User* layer countermeasures.

Indeed, someone may consider a different granularity or categorization of layers and define another set. The important point is that each countermeasure is assigned a layer in which protects the system against a threat.

B. Input Analysis

Given a software application $S = (C, T, G)$, in this section, we analyze its security with respect to our model.

The first step of analyzing our inputs is to calculate the amount of risk that every threat brings to our system. As it is shown in (1), the risk of every threat t is r_t, which is equal to the probability of happening threat multiply in the impact of the threat on the system. We can simply compute the risk of each threat in our model having the threat set T.

$$r_t = p_t * impact_t \tag{1}$$

Next, we calculate d_t or the amount of deterrence against threat t. To calculate d_t we remind that an edge e, which connects a countermeasure c to a threat t is labeled with f showing the coverage of t by c. Assume that a threat t is mitigated by c_i, which covers f_i percentage of the threat. Then a new countermeasure c_j for t with the coverage ratio f_j applies on the part of t which has not been covered by c_i. For example, imagine a system with two countermeasures and one threat. The first countermeasure covers 60% of the threat and the other one covers 40% of the threat. If we implement the first countermeasure, then 40% of the threat has not been covered yet. That is 40% of the threat risk remains. After implementing the second countermeasure, its coverage ratio 40% is applied on the remaining uncovered part. That is, the second countermeasure covers 40% of the remaining risk, not 40% of the total risk. Therefore, the two countermeasures mitigate 76% of the threat risk (60% for the first countermeasure plus 16% for the second one). If we would implement the countermeasures in the reverse order, the same result is obtained regarding the threat mitigation.

According to above intuition, (2) calculates deterrence against a threat t. In this equation, when we want to sum the deterrence of a countermeasure c, d'_t is the amount of deterrence for all countermeasures implemented before c. Obviously, d'_t is a real number in [0..1] initialized by zero.

$$d_t = \sum_{c|\,(c,t,f)\in E} f \times (1 - d'_t) \tag{2}$$

Having in mind that a threat protects the system in a specific level, the amount of deterrence for every layer can be computed individually. We compute $d_{t,l}$ as the deterrence value for threat t belonging to layer l. $d_{t,l}$ is calculated as shown in (3).

$$d_{t,l} = \sum_{(c,l)\in C} d_t \tag{3}$$

Having d_t, the amount of remaining risk for a threat t can be calculated. Remaining risk is the amount of risk without deterrence against it. As we calculated the amount of deterrence for every layer, $d_{t,l}$, the amount of remaining risk can be computed for a layer l, as well. Remaining risk of t in a layer l is shown with $rr_{t,l}$. Equation (4) shows the calculation of $rr_{t,l}$. Since r_t and $d_{t,l}$ are values between zero and one, $rr_{t,l}$ is also a value between zero and one.

$$rr_{t,l} = r_t \times (1 - d_{t,l}) \tag{4}$$

Our model let us compute the real value of implementing a countermeasure. Adding a countermeasure c to the system, effects on the remaining risk. The difference between all the remaining risks before and after implementing c gives us the real value of c. Intuitively, the real value of a countermeasure to be added in the system is completely dependent on the existing countermeasures and how much they provide deterrence against the threats. Equation (5) calculates v_c as the real value of implementing c. In (5) $rr'_{t,l}$ is the remaining risk of t in layer l before implementing c.

$$v_c = \left(\sum_{t\in T} rr_{t,l}\right) - \left(\sum_{t\in T} rr'_{t,l}\right) \tag{5}$$

C. Analyzing security based on the defense-in-depth

After analyzing input of our system, we calculate the amount of defense-in-depth to evaluate security of the system. To calculate defense-in-depth, first we find the amount of defense in each layer. Having the amount of defense in all layers, we can then decide on the amount of defense-in-depth.

Defense in a layer l has a direct relation with the amount of deterrence. Big deterrence value against all the threats in the layer l means that there is adequate defense in this layer. On the other hand, big value for the remaining risk in a layer l means that the defense in this layer is not configured properly. Based on the above discussion, (6) calculates the amount of defense in layer l.

$$Defense_l = \sum_{t \in T} \frac{d_{t,l}}{rr_{t,l}} \quad (6)$$

Now, we compute the total amount of defense-in-depth, and show it with *DiD*. There are different strategies to infer about the total layered defense, based on different intuitions.

Weighted Mean Strategy (WMS): Our suggestion is to use the Weighted Mean Strategy (WMS) to calculate the amount of defense-in-depth. To this purpose each layer l_i is given a weight w_i. The weight w_i is a coefficient for the amount of defense in layer l_i. The defense–in-depth is then computed by the weighted mean of defense values. Equation (7) shows the formula of WMS where w_i is the weight of layer l_i. it is worth to mention that the sum of weights is equal to one ($\sum_{i \in L} w_i = 1$).

$$DiD = \frac{\sum_{i \in L} w_i \times Defense_i}{\sum_{i \in L} w_i} \quad (7)$$

Remark - Choosing weights: the weight of a layer is chosen regarding the importance of defense in the layer. It is also interpreted as the attacker's power regarding each layer. For example, if the attacker situation is inside the organization network, the defense corresponding to the *Network* layer may be meaningless for such an attacker. That is, the weight of *Network* layer can be considered as zero. As another example, for situations that there is no way to change the business logic of an application, the *Business* layer is not important and its weight can be zero. If all the layers have the same importance, $w_i = w_j$ for each l_i and l_j.

In the weakest link thought of security, the weakest layer determines the amount of defense-in-depth. Therefore, we can set the weight of weakest layer equal to one and other weights set to zero.

D. Proposing a countermeasure based on defense-in-depth

After computing defense-in-depth, a decision must be taken by security expert whether security is acceptable or it should be increased. If security with respect to the defense-in-depth is acceptable, no more actions need to be taken. But if we wish to improve the security of our system, we will go to next step which proposes a countermeasure to improve the defense-in-depth. To choose a countermeasure, two parameter are important. First, the cost of implementing the countermeasure. Second, the effect of countermeasure on the amount of defense-in-depth. Obviously, it should minimizes the cost and maximizes the defense-in-depth.

The result of multiplying the real value of a countermeasure, v_c in (5), in the weight of the layer corresponding to the countermeasure is used to find the proposed countermeasure. On the other hand, the principle of *economy of mechanism* says that a mechanism with lower cost is more desirable. Therefore, as it is shown in (8), we compute priority of countermeasures. Finally, the countermeasure with the highest priority is proposed for implementation.

$$P_c = \frac{w_l \times v_c}{cost_c} \quad (8)$$

IV. CASE STUDY

We evaluate our approach on a VOIP software system. The input model consists of 58 threats to the software and 127 corresponding countermeasures. For all layers, there are some countermeasures. Table 1 shows the number of countermeasures in each layer. The threats and countermeasures form a relation consisting of 206 triples (*c, t, f*). This is because a countermeasure may be connected to different threats, each of which has its own coverage ratio. Table 2 shows some selected threats. Also, Table 3 shows a subset of security countermeasures together with their coverage ratio for threats. For example, RTP is an encoding used for sending voice over network. Sniffing RTP packets is a threat, which can be covered by using SRTP protocol instead of the RTP that covers 98% of the risk of sniffing RTP. Using TLS for communication can also cover this threat up to 98%. Due to space limitation it is not possible to bring all threats and countermeasures.

After analyzing input, we calculated the amount of defense-in-depth in this software. Then we proposed a countermeasure to improve security based on the defense-in-depth.

Table 1- The number of security countermeasures in each layer

Physical	*DataAccess*	*Business*	*Network*	*User*
10	12	49	33	23

A. Analyzing input

In the first step, we calculate the risk for every member of T using (1) which gives us a list of $\boldsymbol{r_t}$ (rt is the risk value pertaining to threat t). For example, in Table 2 for thread 1, $\boldsymbol{p_t}$ and $\boldsymbol{impact_t}$ are 90% and 30% respectively. Therefore, $\boldsymbol{r_t}$ for this thread is equal to 0.27. After that, we calculate deterrence against every threat using (3) which also gives us a list of $\boldsymbol{d_{t,l}}$ for all threats. With the given countermeasures in Table 3 against thread 1, we calculate the amount of $\boldsymbol{d_{t,l}}$ for this threat. Then, the remaining risk of every threat ($\boldsymbol{rr_{t,l}}$) can be calculated using (4). In our example, $\boldsymbol{d_{1,Network}}$ is equal to 0.88 and $\boldsymbol{d_{1,DataAccess}}$ is equal to 0.7, $\boldsymbol{rr_{1,DataAccess}}$ is equal to 0.081, and $\boldsymbol{rr_{1,Network}}$ is equal to 0.0324. Now, we have a list of $\boldsymbol{r_t}$, a list of $\boldsymbol{d_{t,l}}$, and $\boldsymbol{rr_{t,l}}$ for every layers, where every list has 58 members(our software has 58 threats and every threat has a $\boldsymbol{r_t}$, $\boldsymbol{d_{t,l}}$ and $\boldsymbol{rr_{t,l}}$. The last step in analyzing the input is calculating the value of every countermeasure by (5) which gives us a list of values with 127 members. For example, for the second countermeasure in Table 3, $\boldsymbol{rr_{1,Network}}$ before implementing this countermeasure is 0.167 and after its implementation is 0.0324. Therefore, $\boldsymbol{v_c}$ for this countermeasure is 0.1346. The value of third countermeasure in Table 3 is calculated and $\boldsymbol{rr_{8,User}}$ before implementing it is 0.16 and after implementing this

countermeasure, it would be 0.144 and value of this countermeasure would be 0.016. This can show us that using firewall can be more valuable rather than teaching users about notification announcement.

Table 2- Some threats in a VOIP system

Threat ID	Threat Name
1	Sending a flood of SIP packets for denial of service
2	Spoofing RTP packets
3	Guessing password of SIP users
4	Users lack of knowledge for security issues
5	Software bad installation and configuration
6	Harassing phone calls or annoying advertising calls
7	Deceiving users by imitating sound or impersonation
8	Spoofing phone identifier from a user in another provider
9	Denial of service due to sending BYE packets
10	Sniffing voice

Table 3 – Sample countermeasures and their relation with threats

Countermeasure	Layer	Threat ID	Coverage Ratio
Using TLS for communication	*Business*	10	90%
Using firewall for protecting network	*Network*	1	80%
Teaching users that notifications are announced only through specific numbers	*User*	8	10%
Creating an announcement process for users	*User*	8	20%
Using IDS to prevent attacks	*Network*	1	40%
Use TCP instead of UDP	*Business*	10	10%
Prepare a security document for users and administrators	*User*	4	80%
Auditing unauthenticated RTP packets	*Network*	2	50%
Restricting extensions to one or a range of IP	*Network*	3	80%
Preparing a document for installation and configuration	*User*	5	80%
Reading the characteristics of voice message sender	*business*	7	90%
Auditing SIP Packets	*DataAccess*	1	70%

B. *Analyze security based on defense-in-depth*

For analyzing security based on defense-in-depth, first we should calculate the amount of defense for every layer by equation (6). We presented the final result of equation (6) in Table 4.

Now we compute the amount of defense-in-depth for our software. To this purpose, we use equation (7). Before putting values inside this formula, we should figure out the weight for every layer. As it was mentioned before, deciding for weight of a layer depends on the analyst's strategy. As an example, we have chosen the weight of each layer as shown in Table 5. We can see in Table 5 that the weakest layer is about five times more important than the strongest layer. In addition, the second weak layer is about four times more important than the strongest layer, and the third weak layer is about three times more important than the strongest layer. Finally, the fourth weak layer is about two times more important than the strongest layer.

Table 4-the result of calculating defense for every layer

layer	amount of defense
Physical	3.6685
DataAccess	0
Business	1073.248
Network	576.0655
User	0.0776

Having the mentioned weights, the amount of defense-in-depth for this software is equal to 140.038, according to (7).

Now, we want to increase the security for our software, so we should go to the next step and propose a countermeasure.

Table 5- our strategy to weight every layer

layer	Weight
Weakest layer	0.34
Second weak layer	0.27
Third weak layer	0.2
Fourth weak layer	0.13
Strongest layer	0.06

C. *Proposing a countermeasure base on defense-in-depth*

To propose a countermeasure for this software, we use (8) and consider the weights assigned to layers in Table 5. After using (8) for all our countermeasures, the highest priority is devoted to a countermeasure in *DataAccess* layer. This countermeasure tells us to use TLS as our default communication service to transmit data between our DBMS and data storage. On the other hand, we see in Table 4 that the amount of defense in *DataAccess* layer is zero, indicating the importance of implementing a security countermeasure for this layer. Also, the threat-countermeasure graph of the system confirms that this countermeasure covers many threats, so reduces the remaining risk in this layer.

Table 6- new amounts of defense for every layer

layer	amount of defense
Physical	3.6685
DataAccess	309.9937
Business	1073.248
Network	576.0655
User	0.0776

After implementing the proposed countermeasure, our input model changes to include the new countermeasure. We have to re-analyze our input regarding the new countermeasure. It means that we should calculate r_t, $d_{t,l}$, and $rr_{t,l}$ for all threats in each layer, and v_c for all countermeasures. Having these values, we are able to calculate the new amount of $Defense_i$ for all layers. Table 6 shows the result of calculations after implementing the new countermeasure. Based on these amounts and given the weights in Table 5, we calculate the new amount of defense-in-depth, which is 202.299.

V. CONCLUSION

In this study, we propose a new model to analyze security with respect to the defense-in-depth principle. To calculate security based on defense-in-depth, we require an input model of software. This model includes three sets. The first set represents existing threats in our software. The second set describes implemented or possible countermeasures for our software. The last set express the relation between countermeasures and threats. We use our input model to analyze our software security. To this purpose, we compute the risk of threats as well as the amount of deterrence against threats, and consequently, the remaining risk for threats. Next, we calculated defense-in-depth, based on the amount of defense in every layer. Our model is able to propose a countermeasure, which maximizes the defense-in-depth while reduces the remaining risks. Using a case study, which is a VOIP system, we show that our model is practical and can help software developers as well as security analysts to find the best countermeasure in order to improve software security with focus on the defense-in-depth principle.

REFERENCES

[1] R.N. Mead, J.H. Allen, S. Barnum, R.J. Ellison, and G.R. McGraw. *Software security engineering: a guide for project managers*. Addison-Wesley Professional, 2004.

[2] J.A Wang, H. Wang, M. Guo, M. Xia. "Security metrics for software systems" *Proceedings of the 47th Annual Southeast Regional Conference, ACM.* Clemson, South Carolina. 2009, Mar 19. PP 47.

[3] P. Mell, K. Scarfone, and S. Romanosky, "A Complete Guide to the Common Vulnerability Scoring System (CVSS), Version 2.0" *Forum of Incident Response and Security Teams*. July 2007.

[4] N.S. Dorri, H.M. Ali, J. Rasool. "Measuring Software Security Using SAN Models". in *Information Security and Cryptology (ISCISC), 2012 9th International ISC Conference on. IEEE*. Tabriz, Iran. PP 80-86. 2012.

[5] G. Stephen, T. Joseph, M. Randy. "Advancing The Defense-in-Depth Model". In *System of Systems Engineering (SoSE), 2012 7th International Conference on. IEEE*. Genova, Italy. pp. 285-290. 2012.

[6] R . Lippmann, K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, R. Cunningham. "Validating and restoring defense in depth using attack graphs". In *Military Communications Conference(MILCOM), IEEE*. Washington, DC, USA. PP 1-10. Oct 23-25 2006.

[7] P. Mell, J. Shook, R. Harang. "Measuring and Improving the Effectiveness of Defense-in-Depth Postures". In *Proceedings of the 2nd Annual Industrial Control System Security Workshop. ACM*. pp. 15-22. 2016, December.

[8] H. Rabiee. "Proposing a Method to Specify and Improve Software Security Level". Theisis for Master of Science. Malek Ashtar Univeersity of Technology. Tehran. Autmn 2014. [persian]