Analyzing Security Aspects during Software Design Phase using Attack-based Analysis Model

Saman Hedayatpour, Nazri Kama and Suriayati Chuprat

Advanced Informations School, Universiti Teknologi Malaysia, International Campus, Kuala Lumpur, 54100, Malaysia
samah.edayatpour@gmail.com, nazrikama@ic.utm.my, suria@ic.utm.my

Abstract

In recent years, concentration on software design phase for evaluating security into the developing software increased where the cost of fixing errors in design level is several times less than the cost of fixing errors in the coding or implementation levels. One of the main challenges in facing current models that evaluate security into the software design phase refers to the need for existence of security experts to analyze the system from a security angle of view while this additional task makes the project more costly and lengthy. In this work we address this problem by defining a method for using known attacks’ and threats’ properties and behaviors instead of using a drawn misuse case for assessing potential risks in the developing software. The main contribution of this work refers to defining a model for analyzing security aspects into the software design phase while additional cost and time are not required for system analyzing by security experts.

Keywords: Secure software, software design, software threats, security analysis

1. Introduction

The media and press are full of news on how software companies have lost annually thousands and millions of dollars, public confidence and markets due to different security breaches and bugs [1]. According to the statistical reports presented by computer security agencies, the number of security incidents in recent years has been increased rapidly [2, 3] while malicious software have been tracked as the main cause in most of these security incidents [4]. Besides the commercial view and due to widespread interconnection of software systems with sensitive data such as finance, military, and public health care, the awareness on emphasizing of risk minimization among the software developers increased [5]. According to the concepts of secure software development, software developers must ensure that the developing software protects system and user assets against possible security threats [6].

Currently a number of different approaches exist for evaluating security of software into the different phases of the software development life cycle. Among these different phases, software design has most significance where the cost of fixing errors is several times less than the cost of fixing errors in the coding or implementation levels [7, 1]. Since the Unified Modeling Language (UML) offers a great opportunity for designing high quality software systems and became such a standard in object oriented software modeling [8], it is used as the platform for placing security features in all security evaluation methods with concentration on software design. Beside the issue of using UML diagrams for placing security features which is an undeniable property of different security evaluation models (working on software design phase), all of the current works ask security experts for analyzing system from a security
angle of view and defining possible threat scenario. In this paper the method of identifying potential risks is the factor used to distinguish between current security evaluation models where each of these models has its own security of view and follows its own concepts for identifying potential risks. The first group of security evaluation models asks for direct existence of security experts over entire of the design phase and risky points will be identified and placed directly by security experts. The second group of security evaluation models tries to gain revealed information from defined threat scenario by using that into their own methodology of risk identification.

Each of these approaches has strong and weak points that make each of them more suitable for using in specific cases of software development. Regardless of which method has better performance; all current works from both groups ask security experts to analyze the system and define scenario of possible threats against different system functionalities. This need for existence of security experts into the project is the main problem to deal with these models where the additional role (role of security experts) and duties (analyzing system and defining threat scenario) make the project more costly and longer [9].

To address this problem, in this work we introduce the Attack-based security analysis model with ability of evaluating security into the designed software without asking security experts for analyzing the system. In this model instead of using threat scenario (defined by security experts), we use known threats’ properties and behaviors for assessing potential threats. As the consequence of eliminating the need for defining threat scenario, additional cost and time are not required in this model for conducting a security analysis by security experts. Through this model, the entire process of Attack-based security analysis will be conducted in two separate sides of rule developer and the client. This process starts in the rule developer side where we have two key notions of “Threats’ properties and behaviors” which refers to the signs, actions, conditions, and everything else that may be used for threat identification, and “Threat grammar” which refers to the set of structural security rules that explains all possible forms of applying security features. The threat grammar will be defined by individual or organized security experts and depend on quality and financial targets of rules’ developer, may be accessible by free in the form of open source or as a commercial package to be sold. With moving to the client side we have also two key notions of “Threat tracer” which is a designed algorithm intended to search for possible threats into the UML diagrams, and “Impact analysis” which refers to the determination level of impact in facing with launching a successful attack. In this work we gain the UML standard diagrams for visual modeling of the system and the UML extension mechanism of stereotype for adding security aspects (noted by Attack-based security analysis) into the UML diagrams of the system. The rest of this paper organized as follows: Section 2 brief introduction in similar works. Section 3 describes the overall structure of the proposed Attack-based security analysis. Section 4 is an evaluation on the efficiency of this model. In Section 5 we discuss on contributions of work and finally Section 6 is conclusion and future work.

2. Related Works

In this section, we provide an overview on current security evaluation models working on the software design phase. The mechanism of addressing security aspects is the factor used to distinguish between current models. Based on this factor, “direct security design” is the first group of current security evaluation models that asks security experts for working in parallel with developers on UML diagrams for placing security features based on defined threat scenario. The second group of current security evaluation models is “indirect security design” that beside the defined threat scenario, use their own methodologies for addressing security points.
2.1. Direct Security Design

In this type of security evaluation models, the role of security experts will not be finished with defining possible threat scenario but also they have to work in parallel with developers for identifying risky points into the UML diagrams and placing required security features into the right places. In this category, the Unified Modeling Language Security (UMLsec) is the most famous work which was proposed [8] in Munich University of Technology of Germany. The general purpose of this model refers to making knowledge of security experts available to software developers by defining attack scenario and placing security features on UML diagrams. The semantic of the UMLsec generates a function in association with each component of the system; this function takes a set of messages and a component state in input where a set of messages and a new component state will be in the output of this function.

The Model-Based Design and Analysis of Permission-Based Security was proposed [10] along similar lines with UMLsec also at Munich University of Technology of Germany. In terms of matching workflows and the design of the security permissions together, this approach was proposed for embedding security permissions into the early phase of software design by using object-oriented concepts in form of UML models and providing consistency checks to supports model based security analysis. These permissions are defined on a certain level of restrictions based on the functionalities which the system is designed for.

2.1. Indirect Security Design

Indirect security design refers to another group of security evaluation mechanisms which role of security experts is limited to defining possible threat scenarios. Instead of asking security experts for working with developers and placing security features, these models use their own defined methodologies and techniques for placing security points.

The UML Base Modeling and Analysis of Security Threats [11] is a UML-based framework for modeling and analyzing security threats. As the main concepts in this approach, potential security threats against the system will be specified by using UML sequence diagrams while for modeling intended system functions, state chart diagrams will be used. These state chart diagrams automatically will be translated into a graph transformation system, which provide a theoretical foundation to simulate the execution of state transitions where a sequence diagram in association with a security threat is interpreted as a sequence of paired graph transformations (the analysis of a security threat is conducted through simulating the state transitions from an initial state to a final state triggered by method invocations).

The verification and Trade-Off Analysis of Security Properties in UML System Models is another work from the category of indirect security design [12]. As the main point, this work uses the Aspect-Oriented Risk-Driven Development (AORDD) methodology for developing secure systems with ability to determine how different security mechanisms work in different situations and parts of the system. This methodology begins with defining system assets by system designers and identifying potential attacks against these assets by security experts. Once a risk was determined, it must be mitigated by incorporating security mechanisms methodically into the system design. Then, the resulting design will be formally evaluated by designers to ensure that the threat has been mitigated (meet other project mechanisms are still allowed).
3. Attack-based Security Analysis

In this paper, the entire process of conducting Attack-based security analysis has been divided into five main steps of pre-process, defining security rules (threats grammar), security evaluation, impact analysis and risk determination. Since in this model we have two operating sides of the security rule developer and the client (software of security analyzer and system designers), each of these steps will be handled in one of these operating sides. To more clarify on borders of operation sides and their included operations, Figure 1 shows the technical structure of Attack-based security analysis model.

![Figure 1. Structure of Attack-based Security Analysis Model](image)

3.1. Pre-process

As a general description for the pre-process step, all information about the target system and the attack must be gathered and formatted into the usable expressions. In this step and through the first task, all possible vulnerabilities and threats against a comprehensive scenario of attack must be identified and listed (performance of this model directly is depend on the number and sensitivity of identified vulnerabilities and threat). Table 1 shows a numbers of identified vulnerabilities and threats in facing with SQL injection [13, 14].

Into the second task, technical points in association with each identified threats and vulnerabilities must be extracted and listed. The output of this task will be gathered and stored technical information related to identified threats and vulnerabilities such as discovered bugs into the specific version of the software or found weaknesses into the security mechanisms. Table 2 represents the gathered technical points in association with the addressed vulnerabilities and threats in Table 1.
Table 1. Example of Gathered Vulnerability/Threat Pairs

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Threat-Source</th>
<th>Threat Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storing critical information such as usernames and without any encryption mechanism</td>
<td>Storing critical information in form of plain text</td>
<td>By launching a successful SQL injection, attacker will has access to read and write the secure data</td>
</tr>
<tr>
<td>SQL Server 2000 allows batch-execution of statements</td>
<td>Using weak versions</td>
<td>SQL Server 2000, allow multiple statements separated by semicolons to be executed at once</td>
</tr>
<tr>
<td>Lack of a mechanism to install patches regularly and timely</td>
<td>Not being updated</td>
<td>Taking advantage from the found and known bugs in certain old versions for inject the SQL query</td>
</tr>
<tr>
<td>Receiving the input from un-trusted users and send it to the data base without validate the content</td>
<td>Lack of Input validation mechanism</td>
<td>Inserting SQL query and command instead of normal input</td>
</tr>
<tr>
<td>Lack of any mechanism for checking contents of error messages</td>
<td>leaking data from error messages</td>
<td>Inserting unexpected input to receive error messages and extracting valuable information, which a error message may Show up</td>
</tr>
</tbody>
</table>

Table 2. Example of Identified Technical Points

<table>
<thead>
<tr>
<th>No</th>
<th>Threat</th>
<th>Basic class</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQL Injection</td>
<td>Input validation</td>
<td>Using white list</td>
</tr>
<tr>
<td>2</td>
<td>SQL Injection</td>
<td>Input validation</td>
<td>Using blacklist</td>
</tr>
<tr>
<td>3</td>
<td>SQL Injection</td>
<td>Input validation</td>
<td>Using prepare statements (bound parameters)</td>
</tr>
<tr>
<td>4</td>
<td>SQL Injection</td>
<td>No leaking data</td>
<td>Checking content of error messages</td>
</tr>
<tr>
<td>5</td>
<td>SQL Injection</td>
<td>Updated versions</td>
<td>Using updated application firewall</td>
</tr>
<tr>
<td>6</td>
<td>SQL Injection</td>
<td>Storing critical data</td>
<td>Store secrets securely (Cryptography)</td>
</tr>
<tr>
<td>7</td>
<td>SQL Injection</td>
<td>Secure version</td>
<td>Using updated secure version of database</td>
</tr>
</tbody>
</table>

3.2. Defining Security Rules (Threat Grammar)

Once all required technical points gathered and placed into the tables, these threat expressions shall be converted into the form of security rules. We define security rules (threat grammar) by applying format of right regular grammar on threat expressions. As an example for defining security rules, the threat grammar in the example case of SQL injection may be defined as follow:

\[
Q^0 \rightarrow nQ^1 | bQ^2 | wQ^3 | pQ^4 | vQ^5
\]

Where

- \(Q^0\): No input validation
- \(Q^1\): Using whitelists
- \(Q^2\): Using blacklists
- \(Q^3\): Using prepare statements
- \(Q^4\): Using version of SQL server 2000
- \(Q^5\): Checking messages’ contents
- \(Q^6\): Store secrets securely
- \(Q^7\): Using updated application firewall

The Attack-based security analysis model uses Nondeterministic Finite Automaton (NFA) to visualize defined threat grammar and determine the likelihood of attack into the current design of developing software. As the first requirement for using NFA, the final states and the status of risk in different final states shall be noted. In case of dealing with mentioned set of security rules, all states except \(Q^0\) are the final states with the status of \(Q^{1,2,3}\) (High), \(Q^{4,5,6}\) (Medium), and \(Q^7\) (Low). In this example case,
presence and priority of used security techniques are the causes for achieving different level in likelihood of attack. Base on this threat grammar, following security sequence are possible:

<table>
<thead>
<tr>
<th>Security Technique</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>No input validation mechanism:</td>
<td>High</td>
</tr>
<tr>
<td>Using SQL server 2000:</td>
<td>High</td>
</tr>
<tr>
<td>Blacklist:</td>
<td>High</td>
</tr>
<tr>
<td>Prepare statements:</td>
<td>High</td>
</tr>
<tr>
<td>White list:</td>
<td>Medium</td>
</tr>
<tr>
<td>Blacklist + prepare statements:</td>
<td>Medium</td>
</tr>
<tr>
<td>Prepare statements + black list:</td>
<td>Medium</td>
</tr>
<tr>
<td>Whitelist + (store secrets securely OR checking messages’ contents ) + application firewall:</td>
<td>Low</td>
</tr>
</tbody>
</table>

The automaton starts from $Q^0$ state and each applied security technique triggers a jump to the new state which may have a different risk level. If the automata stopped at states of $Q^1$, $Q^2$, or $Q^3$, means the level of threat likelihood in the current design of the system is high while if stopped at states $Q^4$ or $Q^5$, the level of threat likelihood will be medium and finally if stopped at state $Q^5$, means the level of threat likelihood in the current design of the system is Low. Figure 2 shows the set of defined security rules in the form of nondeterministic finite automaton where these NFA are defined as ($Q$, $Σ$, $δ$, $q^0$, $F$):

$Q = \{q^0, q^1, q^2, q^3, q^4, q^5, q^6, q^7\}$

$F = \{q^1, q^2, q^3, q^4, q^5, q^6, q^7\}$

$Σ = \{v, n, b, w, p, c, s, f\}$

$q^0 = \{q^0\}$

$δ(q^0, b) = q^1$

$δ(q^0, p) = q^2$

$δ(q^0, v) = q^3$

$δ(q^0, n) = q^3$

$δ(q^0, w) = q^4$

$δ(q^1, b) = q^6$

$δ(q^2, p) = q^6$

$δ(q^3, v) = q^5$

$δ(q^4, (c | s)) = q^5$

$δ(q^5, λ) = q^5$

$δ(q^6, λ) = q^5$

$F = \{q^1, q^2, q^3, q^4, q^5, q^6, q^7\}$

Figure 2. Visualization of Second Rules Set

In parallel with the process of defining set of security rules (regardless of threats’ types), vulnerable places and condition of threat assessment into the different UML diagrams must be identified. In this purpose, security rules’ developers have to specify potentially vulnerable places against target threat and the condition that the threats may be determined. This information (attributes) is attached into the package of security rules and will be used into the process of Attack-based security analyzing.
3.3. Security Evaluation

Once the set of security rules was defined by security rules developers, the process of security evaluation starts with importing these security rules into the security analyzer. Based on security rules’ requirements and attributes, the security analyzer asks for drawing or importing different UML diagrams. As the next operation, the algorithm of threat tracer attempts to determine vulnerable places (base on highlighted condition of assessment into the rules’ attributes) into the different parts of UML diagrams.

As an example for showing this process until here: Figure 3 shows two of required security considerations in facing with SQL injection into the activity diagram of an online registration system. The first security consideration represented by stereotype of <<InputValidation>> that indicates a vulnerable place against the SQL injection attack where input validation mechanisms must be checked. Based on content of Table 2, the “whitelist”, “blacklist”, and “prepare statements” refers to the main input validation mechanisms that may be applied at this point. The stereotype of <<ErrorContentCheck>> is the second highlighted security aspect in Figure 3 and emphasizes on using a mechanism for checking contents of error messages and strip them from information that attacker can gain for launching SQL injection.

![Figure 3. Identified Vulnerable Points and Associated Security Aspects](image)

Checking status of security (applied security techniques and considerations) into the detected vulnerable points is the next task in process of security evaluation. Depend on using security rules, more than one security techniques may be existed for each vulnerability. In such cases, the order and number of used security techniques effects on status of risk in that vulnerable point. Based on rule developers’ advice, applying each technique into the vulnerable points provide different level of protection (applying a combination of security techniques may reduce just one level of risk while applying another single technique may reduce two or more levels of risk in the same place).

Into the last task of security evaluation, the factor of attack likelihood in developing software must be determined. The factor of attack likelihood reflects completeness in
applying required security techniques. As is shown in Figure 2, the level of risk for the entire project will be determined by following the path of used security mechanism into the defined nondeterministic finite automaton. In an example case that defined set of security rules in Figure 2 considered for conducting security evaluation; applying mechanism of whitelist for input validation, using a mechanism for checking content of error messages or using an encryption technique for protecting secure data, and using an updated firewall may reduce level of risk from of High to Low. Figure 4 shows how the likelihood of risk was determined in this case by following the path of applied security techniques into the vulnerable points.

![Figure 4. Visualization of First Set of Rules](image)

This automaton started from Q₀ state and by receiving input of w (whitelist) and using rule of Q₀ → wQ₁ jumped to Q₁ state. In continue, receiving inputs of s or c (store critical data securely or error content checking) and f (updated firewall) respectively trigger jumps to Q₅ state and Q₃ state by using rules of Q₁ → s, cQ₅ and Q₃ → fQ₇. Since the automaton stopped in Q₇, this state will be the final state and its status of risk level demonstrates the likelihood of attack in this example case.

### 3.4. Impact Analysis

After determination of attack likelihood in the security design of the target system, the impact of launching a successful attack is the second require factor which must be determined to be used in calculation of risk level [15]. As the main task in conducting impact analysis, the necessary information on system mission, system and data criticality, and system and data sensitivity must be gathered.

The mission impact analysis report and the asset criticality assessment report refer to two of the most significant organizational documentations that may be used for obtaining required information. A mission impact analysis report identifies the level of impact in case of compromised organization’s assets and an asset criticality assessment report addresses the sensitive assets such as software, hardware, systems, technology, and services into the organization [13].

If these reports were not exist, the required level of protection to assure system and data’s availability, integrity, and confidentiality can be used to determine the system and data sensitivity. Regardless of how sensitive is the IT system and its data, owners of the system are the only ones who are responsible to determine the level of impact for the system and information. Since the most important impacts such as loss of credibility and loss of public confidence cannot be measured in specific units, they will be described in terms of high, medium, and low impacts.

### 3.5. Risk Determination

Through the previous steps and base on basic elements such as threat capability, motivation, nature of the vulnerability, and effectiveness of current controls; two factors of attack likelihood and impact of launching a successful attack were determined. The purpose of this step refers to using these two determined factors to
calculate the level of risk into the developing software system. According to the method of risk measurement introduced by the National Institute of Standards and Technology SP800-30 risk management guide [13], the level of risk is derived by multiplying the threat impact level and ratings of the threat likelihood as follow:

\[
\text{Risk Level} = \text{Threat Impact} \times \text{Threat Likelihood}
\]

Since each of these factors (attack impact and threat likelihood) represented by number of scales and levels, a matrix must be developed to show different forms of multiplication between these factors. Table 3 shows the standard matrix for calculating the risk level which has three columns for showing levels of threat likelihood (High, Medium, and Low) and three rows for showing levels of threat impact (High, Medium, and Low). Depend on needs and requirements, the risk level matrix may be defined in different number of rows and columns.

<table>
<thead>
<tr>
<th>Levels of Risk</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low (10)</td>
</tr>
<tr>
<td>Threat Likelihood</td>
<td>Low</td>
</tr>
<tr>
<td>High (1.0)</td>
<td>10 X 1.0 =10</td>
</tr>
<tr>
<td>Medium (0.5)</td>
<td>10 X 0.5 =5</td>
</tr>
<tr>
<td>Low (0.1)</td>
<td>10 X 0.1 =1</td>
</tr>
</tbody>
</table>

4. Evaluation

In this section we evaluate the efficiency of this work by calculating required cost and time for conducting Attack-based security analysis model and compare it with the required cost and time for other security evaluation models from both approaches of direct and indirect security design. The amount of required additional effort (cost and time) for applying each security evaluation model is the factor that must be measured to evaluate the efficiency.

To evaluate the amount of required additional effort (cost and time) for applying each security evaluation model, we use model of COSECMO [16] which is a security extension of COCOMO II [17] and estimates the required effort to assure the security into the developing software system. The COSECMO introduce a new cost driver for security assurance activities, method of estimating the size of security functions, and guidance on setting levels of existing drivers in COCOMO that are affected by the additional requirements for secure systems.

In this cost estimation procedure, we show the required effort (cost and time) for applying each security model by the factor of Person-Months [18] where the average of required effort for producing 1,000 source lines of code (1,000 SLOC = 1 KSLOC) in COSECMO model is 4 Person-Months (PM). Another used factor in this model refers to Evaluation Assurance Level (EAL) that represents the percentage of additional effort required for applying each security model. Table 4 demonstrates the percentage of additional effort required to evaluate 5, 20, and 100 KSLOC of infrastructure software from EAL 1 to EAL 7.
Table 4. Percent of additional effort based on evaluated assurance level & size

<table>
<thead>
<tr>
<th>Size</th>
<th>EAL 1-2</th>
<th>EAL 3</th>
<th>EAL 4</th>
<th>EAL 5</th>
<th>EAL 6</th>
<th>EAL 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>0%</td>
<td>20%</td>
<td>50%</td>
<td>125%</td>
<td>313%</td>
<td>781%</td>
</tr>
<tr>
<td>20,000</td>
<td>0%</td>
<td>40%</td>
<td>100%</td>
<td>250%</td>
<td>625%</td>
<td>1563%</td>
</tr>
<tr>
<td>1000,000</td>
<td>0%</td>
<td>60%</td>
<td>150%</td>
<td>375%</td>
<td>938%</td>
<td>2344%</td>
</tr>
</tbody>
</table>

In terms of requirements for analyzing the efficiency of different security analysis models, the COSECMO uses the following formulas to calculate the required effort to assure the security into the developing software system:

\[
\begin{align*}
\text{Cost (Total Assured)} &= \text{Cost (Assured Devel)} + \text{Cost (Independent Assurance)} \\
\text{Effort (Assured Devel)} &= \text{Effort (Assured EC)} \times 1.18 \\
\text{Effort (Assured EC)} &= \text{Effort (EC)} + \text{Effort (Internal Assurance)} \\
\text{Effort (Internal Assurance)} &= \text{Effort (EC)} \times \%\text{Effort (AL)}
\end{align*}
\]

Where:
- AL: Assurance Level
- Effort (Assured Devel): Effort in person-months from the start to the end of development life-cycle for an assured, secure software system.
- Effort (Assured EC): Effort in person-months for elaboration & construction phases for an assured, secure software system.
- Effort (Internal Assurance): The required additional effort for assuring security into the developing software system.
- Cost (Independent Assurance): The additional cost for asking an independent security agent to verify the developing software system.
- \%\text{Effort (AL)}: Percentage of required additional effort for assuring security into the developing software system.

Since the average of the required effort to produce 1,000 (1 KSLOC) source lines of code in COSECMO model is 4 Person-Months, if in this case we have 5000 lines of code, the base effort would be 20 Person-Months. With the assumption that independent assurance is not required, the total cost in our approach, direct security design, and indirect security design will be calculated as follow:

**Attack-based security analysis model:** Since in our model the task of security analysis by security experts is not required then the additional assurance effort would be zero. In this case the total cost will be calculated as follow:

\[
\begin{align*}
\text{Effort (Internal Assurance)} &= 20\text{PM} \times 0 = 0\text{PM} \\
\text{Effort (Assured EC)} &= 20\text{PM} + 0\text{PM} = 20\text{PM} \\
\text{Effort (Assured Devel)} &= 20 \times 1.18 = 23.6 \\
\text{Cost (Total Assured)} &= 23.6 + 0 = 23.6
\end{align*}
\]

**Indirect security design models:** Since in these models the task of security analysis by security experts is required, the percent of additional effort will be at least in EAL3. In this case the total cost will be calculated as follow: (EAL3 is the minimum of required
effort into this approach where depend on different design of models, EAL may be in range of EAL 3 until EAL 7)

\[
\begin{align*}
\text{Effort (Internal Assurance)} &= 20\text{PM} \times (20\% \times 2.5^{(3-3)}) \\
&= 4\text{PM} \\
\text{Effort (Assured EC)} &= 20\text{PM} + 4\text{PM} = 24\text{PM} \\
\text{Effort (Assured Devel)} &= 24 \times 1.18 = 28.32 \\
\text{Cost (Total Assured)} &= 28.32 + 0 = 28.32
\end{align*}
\]

**Direct security design models:** Beside the task of security analysis by security experts that is required, these models ask security experts for working in parallel with developers for identifying risky points into the UML diagrams and placing security features into right places. In these models, the percent of additional effort will be at least in EAL4 and the total cost will be calculated as follow: (EAL4 is the minimum of required effort into this approach where depend on different design of models, EAL may be in range of EAL 4 until EAL 7).

\[
\begin{align*}
\text{Effort (Internal Assurance)} &= 20\text{PM} \times (20\% \times 2.5^{(4-3)}) \\
&= 10\text{PM} \\
\text{Effort (Assured EC)} &= 20\text{PM} + 10\text{PM} = 30\text{PM} \\
\text{Effort (Assured Devel)} &= 30 \times 1.18 = 35.4 \\
\text{Cost (Total Assured)} &= 35.4 + 0 = 35.4
\end{align*}
\]

Figure 5 shows the cost of applying Attack-based security analysis model, direct security design models and indirect security design models into the four systems with size of 5000, 20000, 100000, and 500000 lines of codes.

![Figure 5. Required Effort based on Evaluated Assurance Level & Size](image-url)
5. Discussion

Based on our achievement from the evaluation Section, the required additional effort for internal assurance is the main factor through the COSECMO security cost estimation model that makes the differences in the amount of required cost for applying each approach of security evaluations. This factor directly addresses the required additional cost and time which are asked by security experts for analyzing the system, defining misuse case of the system, etc.

The result of applying COSECMO security cost estimation model on Attack-based security analysis model and two approaches of direct security design and indirect security design shows that the cost of using this model for evaluating security into the developing software system with sizes of 5, 20, 100 KSLOC is at least 4.72, 18.88, and 94.4 Person-Months less than indirect security design models and 11.8, 47.2, and 236 Person-Months less than direct security design models.

This amount of reduction in required cost and time for evaluating security into the developing software is quite sufficient to show the efficiency of the Attack-based security analysis model for being subjected on various size of developing software. Using this model shall be the only choice for small businesses that cannot afford the expensive cost of using other security evaluation models.

6. Conclusion and Future Work

Since lack of a practically usable mechanism for checking security aspects into the early phase of software design and quick response is the main reason for some things happens with most security vulnerabilities in staying undiscovered [19], this paper introduces a new model for evaluating security aspects into the software design phase where additional cost and time are not required for system analyzing by security experts. The main difference between this model and current models refers to the method of assessing and addressing possible security threats where instead of asking security experts for analyzing system and defining misuse case; known threats’ properties and behaviors are used for assessing and addressing potential threats. The result of experiment on required additional effort for applying different security evaluation models shows the required additional cost and time for applying Attack-based security analysis model is significantly less than current security evaluation models from both approaches of direct security design and indirect security design. Therefore, we would like to infer a statement that this model is the only applicable work for applying in all ranges of software development projects where using current security evaluation models is not reasonable in small businesses (because of the big amount of required cost for applying these models). Since the set of security rules may be defined and broadcasted by unknown individuals or organizations, the main limitation in using the Attack-based security analysis model refers to the need for a mechanism that authenticates the validity of security rules. Lack of sufficient public information on some security threats required for defining security rules, is the second limitation in dealing with this model. As the future work, we intend to continue the research by focusing on other possible platforms beside the UML that may be used for evaluating security aspects into the developing software system. Working on other possible method for defining security rules may be another area for future works.
References


Authors

Saman Hedayatpour, he received his Bachelor Degree in Software Engineering from the Jahad Daneshgahi Institute of Higher Education Iran 2008 and Master Degree of Information Security from the Universiti Teknologi Malaysia 2012. He is currently a PhD student and works on Secure Software Development under supervision of Dr. Mohammad Nazri Kama and Dr. Suriayati Chuprat at the Advanced Informations School, Universiti Teknologi Malaysia.
Nazri Kama, he received his Master Degree in Real-time Software Engineering and Bachelor Degree in Management Information System from Universiti Teknologi Malaysia in 2002 and 2000 respectively. He then obtained his PhD degree at The University of Western Australia in 2011. His research interests are in software development, software maintenance, impact analysis, traceability, and requirement interactions.

Suriayati Chuprat, she received her BSc Computer Science, MSc Computer Science (Real-time Software Engineering) from the Universiti Teknologi Malaysia in 1995 and 2000 respectively. In 2009, she received her PhD in Mathematics from Universiti Teknologi Malaysia and completed several research attachments at the University of North Carolina, Chapel Hill, USA. She is currently a senior lecturer at Advanced Informatics School of Universiti Teknologi Malaysia. Her research interests include real-time scheduling theory, scheduling and resource allocation, parallel and distributed computing, information security and cloud computing.