A Methodology for Capturing Software Systems Security Requirements

Hassan EL-Hadary
Supervised by: Prof. Sherif EL-Kassas
Outline

• Introduction to security
  – Software Security
  – Security Definitions

• Security Requirement Engineering

• Problem Statement

• Survey approaches for eliciting security requirements

• Thesis Objective and Approach
Software Security

• Why secure software?
  – Attacker hacks software systems
  – Vulnerabilities exploited
  – Cost of attack
Secure Software Development

- Developing secure software systems

An example for secure software development lifecycle [1]
Security Definitions

- Assets
- Threats
- Security Concerns
  - Confidentiality
  - Integrity
  - Availability
- Security goals
- Vulnerability
- Countermeasures
Security Definitions

Security Threats, Requirements, and Mechanisms relations [2]
Security requirements

Definition:

• **How** can we achieve security
• **What** need to be secured
• **What must not** occur to achieve security
Security Requirement Engineering

- Consider security **early**
- Update **requirement phase** to support security
- Perform **Security Requirements Engineering**
  - **Elicit**, Analyze, and Validate
Problem Statement

• Elicit adequate security requirements

• Assist non-security experts
Threat Modeling

Threat Modeling for Eliciting Security Requirements [3]
Threat Modeling

• Attack Trees

```
Attack tree [5]
```

```
Open Safe

Pick Lock
Learn Combo
Cut Open Safe
Install Improperly

Find Written Combo
Get Combo From Target

Threaten
Blackmail
Eavesdrop
Bribe

P = Possible
I = Impossible

Listen to Conversation
Get Target to State Combo

and
```
## Threat Modeling
(Threats Classification – STRIDE [6])

<table>
<thead>
<tr>
<th>Threat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Spoofing</strong></td>
<td>access is gained to inaccessible assets using someone else’s credentials.</td>
</tr>
<tr>
<td><strong>Tampering</strong></td>
<td>Occurs when data is changed when an attack is performed.</td>
</tr>
<tr>
<td><strong>Repudiation</strong></td>
<td>a user denies performing an action, but the system has no way to prove an action on a user although the user has performed it.</td>
</tr>
<tr>
<td><strong>Information disclosure</strong></td>
<td>information is disclosed to a user not permitted to see it.</td>
</tr>
<tr>
<td><strong>Denial of service</strong></td>
<td>a valid users become unable to access resources.</td>
</tr>
<tr>
<td><strong>Elevation of privilege</strong></td>
<td>a privileged status is gained by an unprivileged user.</td>
</tr>
</tbody>
</table>
## Threat Modeling
(Threats Ranking – DREAD [6])

<table>
<thead>
<tr>
<th>Damage Potential</th>
<th>The cost of the damage when the threat is exploited.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reproducibility</td>
<td>The rate of reproducing the exploited threat</td>
</tr>
<tr>
<td>Exploitability</td>
<td>The level of skill needed to exploit this threat</td>
</tr>
<tr>
<td>Affected Users</td>
<td>The number of affected users</td>
</tr>
<tr>
<td>Discoverability</td>
<td>The easiness of discovering this threat</td>
</tr>
</tbody>
</table>
Threat Modeling

Steps for eliciting Security Requirements
1. System characterization
2. Assets and access points identification
3. Threats identification (STRIDE)
   - Threat Catalogs
4. Threats Analysis (DREAD)
5. Elicit Security Requirement by negating threats
Threat Modeling

• Expressing security requirements as negative statements is inadequate

• Negative statements should be converted to positive statements
Misuse Cases

Example misuse case [7]
Misuse Cases (Security Use Cases)

Security Team

develops

Misuse Cases
analyze and specify

drive

Security Use Cases
analyze and specify

Assets and Services
are vulnerable to

Security Threats
necessitate

Security Requirements
require

Security Mechanisms

counter

Misuse case with Security Use case [2]
### Misuse cases
(Using generic templates)

<table>
<thead>
<tr>
<th><strong>Generic Misuse Case: Spoof User Access</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary:</strong> The misuser successfully makes the system (physical / human / computerized) believe he is a legitimate user, thus gaining access to a restricted system / service / resource / building.</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
</tr>
<tr>
<td>1) The misuser has a legitimate user’s valid means to identify and authenticate <strong>OR</strong></td>
</tr>
<tr>
<td>2) The misuser has invalid means to identify and authenticate, but so similar to valid means that the system is unable to distinguish (even if operating at its normal capabilities) <strong>OR</strong></td>
</tr>
<tr>
<td>3) The system is corrupted to accept means of identification and authentication that would normally have been rejected. The misuser may previously have performed misuse case “Tamper with system” to corrupt the system.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Misuser interactions</strong></th>
<th><strong>System interactions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Request access / service</td>
<td>Request identification and authentication</td>
</tr>
<tr>
<td>Misidentify and misauthenticate</td>
<td>Grant access / provide service</td>
</tr>
</tbody>
</table>

| **Postconditions:** |
| 1) The misuser can do anything the legitimate user could have done within one access session **AND** |
| 2) In the system’s log (if any), it will appear that the system was accessed by the legitimate user. |
## Misuse cases
(Using generic templates)

### Generic Security Use Case: Access Control

**Path name:** Reject invalid authentication

<table>
<thead>
<tr>
<th>Misuser Interactions</th>
<th>System Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>System Interactions</strong></td>
</tr>
<tr>
<td></td>
<td>Request user identity and authentication.</td>
</tr>
<tr>
<td>Provide valid user id but invalid authentication.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reject misuser by cancelling transaction.</td>
</tr>
</tbody>
</table>

**Postconditions:**
1) Misuser has valid means of user identification but invalid means of user authentication **AND**
2) Misuser not authenticated, not granted access **AND**
3) Access control failure registered.
Misuse cases

• Advantages:
  – Lightweight approach
  – Model interaction between threats and system

• Disadvantages:
  – Informal causing ambiguity
Goal Oriented Requirement Engineering (KAOS)

KAOS Approach [13]
Goal Oriented Requirement Engineering (KAOS)

• Goal Graph

KAOS Goal Graph [13]
Secure Goal Oriented Requirement Engineering (KAOS)

- Anti-Models
Secure Goal Oriented Requirement Engineering (KAOS)

- Construct high-level security goals
  - Example: Avoid[SensitiveInfoKnownByUnauthorizedAgent]
- Build anti-models to elaborate vulnerabilities
- Derive countermeasures and security requirements
  - Avoid[vulnerability]
Secure Goal Oriented Requirement Engineering (KAOS)

- Attack pattern

DOS Attack pattern [14]
Secure Goal Oriented Requirement Engineering (KAOS)

- Derivation of precise security requirements from high-level goals
- Formal methodology enabling automatic processing
Secure Agent orient based approaches (i*)

• Dependency Diagram:
Secure Agent orient based approaches (i*)

• Attacker Analysis:
  “who is likely to attack the system? By what means might a specific attacker attack the system?”

• Dependency vulnerability analysis:
  “Which dependency relationships are vulnerable to attack?”, “What are the chain effects if one dependency link is compromised?”

• Countermeasure Analysis:
  “how to protect the system the attackers exploiting the vulnerabilities “
Secure Agent orient based approaches (i*)

- Adapting security to i* [8]

Diagram:

1. Actor Identification
2. Goal/Task Identification
3. Vulnerability Analysis
4. Attacking Measure Identification
5. Countermeasure Identification
6. Malicious Intent Identification
7. Dependency Identification
8. Attacker Identification
Secure Agent orient based approaches

Advantages:
• Considering the human factor which has a significant impact on security.
  For example, threats may involve tricking other people (doctors or nurses in the case of medical records) to break security goals

Disadvantage:
• Concentrates on confidentiality concerns
Requirement Engineering Using Problem Frames

- Generalized Problem Diagram [9]
Problem Frames (Problem Decomposition)

• Problem decomposition into subproblems [10]

Problem Context

Problem Diagram for Employee Display Information subproblem
Abuse Frames

• Generic Abuse Frame [12]

Model and analyze threats
• Bound scope of security problems
  – What attack harms what asset in what subproblem
Abuse Frames

Abuse Frame for Regime editing subproblem [12]

c: The editing commands that are entered by the Operator.
d: The edit operations performed by the Regime Editor.
e: The effects of the edits on the Light Regime
Security Requirement Engineering based on Problem Frames

1- Model system using Problem Frames
2- Identifying assets and threat descriptions.
   – “What harm could come from violating the [insert security concerns] of [insert asset]?”
3- Identify high-level security goals that avoids threats Example, “Provide data to authorized users”
Security Requirement Engineering based on Problem Frames

4- Constrain Functional requirements to avoid threats

Problem Frame Diagram for constrained functional requirement [11]
Security Requirement Engineering based on Problem Frames

5- Validate security requirements using security arguments

- Trust assumptions
- (domain behavior premises) |- (security requirements)

A |- B means B can be proved from A.
Security Requirement Engineering based on Problem Frames

Advantages:

• Specifies security requirements precisely by constraining functional requirements

• Integrate system behavior with threats using problem frames
Security Problem Frames:

Generic patterns for security problems [15]

SPF: Confidential Data Transmission

Frame diagram

Security requirement (SR)

Malicious subject should not able to derive Sent data and Received data using Transmitted data.

Declarations

\[ y_1 : Y_1; e_2 : E_2; sd : SentData; rd : ReceivedData; td : TransmittedData; ms : MaliciousSubject \]

Preconditions

\[ \{ (ms, sd), (ms, rd) \} \cap known = \emptyset \]
\[ \{ y_1, e_2 \} \subseteq \text{conf} \]

Postconditions

\[ \{ (ms, sd), (ms, rd) \} \cap known = \emptyset \]
\[ \{ (ms, sd, td), (ms, rd, td) \} \subseteq \text{conf} \]

Related

SPF: Integrity-preserving Data Transmission
SPF: Confidential Data Transmission
Thesis Objective

Develop a methodology for eliciting security requirement that adapts problem frames, abuse frames and security problem frames to achieve the two main objectives:

1. Eliciting adequate security requirements
2. Assisting security inexperienced analysts to elicit security requirements
Proposed Methodology

- **Proposed Methodology**

  - Model the system context and functional requirements
  - Identify the system assets
  - Identify threats using abuse frames
  - Crosscut threats with Assets to identify vulnerabilities
  - Constrain Functional Requirements To Obtain Security requirements
  - Instantiate Security Problem Frame
  - Search Security Problem Frames (SPF) Catalog

  - [No Vulnerability]
  - [Found Vulnerability]
  - [If found SPF]
  - [If no results]
Methodology Advantages

- Utilizes different methodologies based on problem frames that complement each other

- Assist the analyst during security requirement elicitation
Contribution

• Interfacing between the Security Problem Frames, Abuse Frames and Problem Frames

• Provide a tool that helps in choosing and instantiating the patterns in the Security problem frame catalog
References


References


References


References


References


References